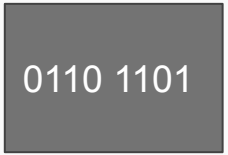
Python Binary File IO

What is it and why do we do it?

What is Binary File IO?

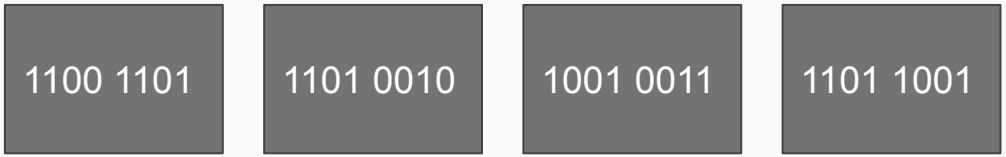
What is binary?

- Binary refers to working with bits 1’s or 0’s
- Bytes are a collection of 8 bits:
  - Ex: 0110 0011
- I shall refer to bytes as a rectangle like this:
  - Pretend the bits are inside from here on out
- Bytes make up the building blocks of Binary IO



What is a file really?

- A file on your computer is nothing but a contiguous sequence of bytes
  - Ex:



- All files, regardless of whether they are text or binary files, look like this.
- So how do we read files? This is where the difference between text and binary files comes in...
  - 1100110 11010110 1101
  -

What is the difference between binary and normal text files?

Text Files:

- Text files assume every byte in a file translates to a character (If ASCII or most of UTF-8)

Binary Files:

- Binary files assume a custom format to how the bytes are laid out
- The first byte may be a character, while the next 4 bytes may refer to a integer number

Data Types Overview

- Python conceals an important albeit tedious programming mechanic known as **Type Declaration**
- The compiled languages (Java, C++ etc) require you to specify the data type so it knows how much memory to reserve for your variable.
  - Ex:
    int a = 5;

Typical Data Type Sizes

- Char (Character Type) **1 Byte**

- Int (Integer Type) **4 Bytes**

- Float (Floating Point Type) **4 Bytes**

- Double (Double- Precision FP Type) **8 Bytes**

- Boolean (Boolean Type) **1 Byte**

Binary file IO

- So how do we read binary files?
  - You must specify a format for binary files
    - I.e The first 4 bytes are for an integer, the next 8 are for a double, and the last two are two individual characters
  - If you don't know the format of the binary file, you'll never make sense of the bytes, and hence can't read from it
  - Text files assume every byte is a character, so if you write a binary file and interpret it as a text file, it wont make any sense!
    - Try opening a .docx file in a basic text editor...

Reading binary files in Python

- In Python, we use the **struct** module to read and write bytes
  - We will use the **pack** and **unpack** functions in this module
- These functions both take a **format string** to specify the format of the bytes to be read or write
- The pack function will then take a sequence of arguments to “pack” into a **bytes** object
  - A bytes object represents a sequence of bytes, similar to a file!
- The unpack function will then take a **bytes** object and translate it based on the format string you provide to the form.

Example 1

A basic example of packing/unpacking three integers:

```
>>> from struct import *
>>> pack('hhl', 1, 2, 3)
b'\x00\x01\x00\x02\x00\x00\x00\x03'
>>> unpack('hhl', b'\x00\x01\x00\x02\x00\x00\x00\x03')
(1, 2, 3)
>>> calcsize('hhl')
8
```

https://docs.python.org/3/library/struct.html

How do I make the format string?

Follow the documentation provided by the Python struct module

Format	C Type	Python type	Standard size	Notes
x	pad byte	no value		
c	char	bytes of length 1	1	
b	signed char	integer	1	(1),(3)
B	unsigned char	integer	1	(3)
?	_Bool	bool	1	(1)
h	short	integer	2	(3)
H	unsigned short	integer	2	(3)
i	int	integer	4	(3)
I	unsigned int	integer	4	(3)
l	long	integer	4	(3)
L	unsigned long	integer	4	(3)
q	long long	integer	8	(2), (3)
Q	unsigned long long	integer	8	(2), (3)
n	ssize_t	integer		(4)
N	size_t	integer		(4)
e	(7)	float	2	(5)
f	float	float	4	(5)
d	double	float	8	(5)
s	char[]	bytes		
p	char[]	bytes		
P	void *	integer		(6)

Reading and Writing Files

- In order to read and write to files in Python, we must first **open** the file
- For binary files we need to add the **binary flag** ‘b’
- After that, we can simply read and write our new **bytes** objects

Example 2

- Things to note:
  - ‘hh’ == ‘2h’
  - For strings, you need to specify the length of the string before the s, hence ‘5s’
  - To read and write files, we need to add the ‘wb’ or ‘rb’ binary flags
  - When seeking, make sure to remember we start at 0 index

```
"""A demo of binary file IO."""

from struct import pack, unpack

# make some bytes!
b = pack('hhf5s', 1, 2, 3.2, 'hello'.encode('utf-8'))

with open('test.dat', 'wb') as file:
    # write our bytes to the file
    file.write(b)

# read in all the bytes of the file at once and decode
with open('test.dat', 'rb') as file:

    bytes = file.read()
    data = unpack('2hf5s', bytes)
    print(data)

# maybe we need to read in only a bit at a time
with open('test.dat', 'rb') as file:
    # read the first 4 bytes of the file and convert
    bytes = file.read(4)
    data = unpack('2h', bytes)
    print(data)
    # jump to the 5th byte and read the next 4 bytes and convert
    file.seek(4)
    bytes = file.read(4)
    data = unpack('f', bytes)
    print(data)
    # jump to the 9th byte and read the next 5 bytes and convert
    file.seek(8)
    bytes = file.read(5)
    data = unpack('5s', bytes)
    print(data)
```

Why use binary files?

- Binary files use much less space than a text files do
  - Ex: to store the number 3.1459265359 would require 12 bytes to store in a text file, but only 4 if a binary file was used!
- They can be processed quicker
  - All text files are interpreted as strings, so if you want numbers out of them you have to convert, but bytes can be converted directly