FREE EDITION · NOTES + 3 SAMPLE MCQS

CUET · COMPUTER SCIENCE · CLASS XII · CODE 308

File Handling in Python

CUET unit: File Handling in Python

By **UniDrill** · NCERT-grounded study material

WWW.UNIDRILL.IN

📌 Snapshot

- Persistent storage (files) is needed because variables lose their values after program execution ends.
- There are two fundamental file types — text and binary — and each stores data internally using byte sequences.
- A file in Python has a complete lifecycle: open → read/write → seek/tell → close, using the `io` module's built-in functions.
- The `with` clause is a safer, auto-closing alternative to explicit `open()` / `close()` calls.
- The `pickle` module serializes and de-serializes Python objects (pickling/unpickling) into binary files — a topic NTA has targeted with direct and application-based questions.

📖 Detailed Notes

2.1 Core concepts

- **What is a file?** A file is a named location on a secondary storage media where data are permanently stored for later access. Programs written in script mode are stored with a `.py` extension; data output can also be stored permanently in a file. (NCERT §2.1, p. 19)
- **Why files are needed:** Variables in a program have a lifetime only till the program is under execution. To reuse input data or generated output (e.g., employee information, inventory, sales), data must be stored permanently on secondary storage devices. (NCERT §2.1, p. 19)
- **Text files:** A text file is a sequence of characters consisting of alphabets, numbers, and other special symbols. Extensions like `.txt`, `.py`, `.csv` are examples. Internally, characters are stored as bytes using ASCII, UNICODE, or another encoding scheme. Each line is terminated by a special End of Line (EOL) character; in Python the default EOL is the newline character `\n`. (NCERT §2.2.1, p. 20)
- **Binary files:** Binary files are stored in terms of bytes (0s and 1s) that represent actual content such as images, audio, video, compressed files, and executable files. They are not human-readable; opening one in a text editor yields garbage values. A single bit change can corrupt a binary file. Specific software is required to read or write binary file contents. (NCERT §2.2.2, p. 21)

- **Opening a file — `open()` :** Syntax is `file_object = open(file_name, access_mode)`. The function returns a file handle stored in `file_object`. The `file_object` has attributes: `<file.closed>` (True if closed), `<file.mode>` (access mode), `<file.name>` (file name). If the file does not exist, `open()` in write mode creates a new empty file. (NCERT §2.3.1, p. 21–22)
- **File access modes (Table 2.1):** Key modes — `r` (read-only, offset at beginning), `rb` (binary read-only), `r+` or `+r` (read and write, beginning), `w` (write, overwrites existing content or creates new file), `wb+` or `+wb` (read, write, binary), `a` (append, offset at end; creates file if absent), `a+` or `+a` (append and read, offset at end). Default mode is read (`r`); default file type is text. (NCERT §2.3.1, p. 22)
- **Closing a file — `close()` :** Syntax: `file_object.close()`. Frees memory allocated to the file. Python flushes any unwritten/unsaved data to the file before closing. If a file object is re-assigned to another file, the previous file is automatically closed. (NCERT §2.3.2, p. 23)
- `with` **clause:** Syntax: `with open(file_name, access_mode) as file_object:`. The file is closed automatically when control exits the `with` block, even if an exception occurs. No explicit `close()` is needed. (NCERT §2.3.3, p. 23)
- **Writing — `write()` :** Takes a single string argument and writes it to the text file. Returns the number of characters written. Numeric data must be converted to string using `str()` before writing. A `\n` must be added manually to mark end of line. `write()` actually writes data onto a buffer; the buffer is moved to permanent storage only when `close()` is called (or `flush()` is explicitly invoked). (NCERT §2.4.1, p. 24)
- **Writing — `writelines()` :** Writes a sequence of strings (from an iterable such as a list or tuple). Does not return the number of characters written. Does not add newline characters automatically. (NCERT §2.4.2, p. 24–25)
- **Reading — `read([n])` :** Reads and returns `n` bytes from the file. If no argument or a negative number is given, the entire file content is read and returned as a single string. (NCERT §2.5.1, p. 25)
- **Reading — `readline([n])` :** Reads one complete line (up to and including `\n`) from the file. If `n` is specified, reads at most `n` bytes from that line. Returns an empty string at EOF. Useful for iterating line by line with a loop. (NCERT §2.5.2, p. 26)
- **Reading — `readlines()` :** Reads all lines from the file and returns them as a list of strings, each ending with `\n`. `split()` separates words within a line; `splitlines()` returns each line as a list element without the newline character. (NCERT §2.5.3, p. 26–27)
- `tell()` **:** Returns an integer giving the current byte position of the file object from the beginning of the file. Syntax: `file_object.tell()`. (NCERT §2.6.1, p. 28)
- `seek()` **:** Positions the file object at a specified location. Syntax: `file_object.seek(offset [, reference_point])`. `reference_point` values: `0` =

beginning (default), `1` = current position, `2` = end of file. `offset` is the number of bytes to move. (NCERT §2.6.2, p. 28)

- **Creating and traversing a text file (§2.7):** Combining `open()` in `w+` mode, `write()`, `seek(0)`, and `read()` allows a single program to both write and read data. Using `readline()` in a `while` loop traverses the file line by line until an empty string (EOF) is returned. (NCERT §2.7, p. 29–31)
- **Pickle module — serialization:** Python's `pickle` module serializes (pickles) any Python object (list, tuple, dictionary, etc.) into a byte stream that can be stored in a binary file or sent over a network. The reverse process — converting the byte stream back to a Python object — is called de-serialization or unpickling. The `pickle` module must be imported explicitly. (NCERT §2.8, p. 32)
- `dump()`: Syntax: `pickle.dump(data_object, file_object)`. Converts (pickles) a Python object and writes it to a binary file opened in `wb` or `ab` mode. (NCERT §2.8.1, p. 32)
- `load()`: Syntax: `store_object = pickle.load(file_object)`. Reads (unpickles) a Python object from a binary file opened in `rb` mode. At end-of-file, raises `EOFError`, which can be caught with a `try…except` block. (NCERT §2.8.2, p. 33)

2.2 Definitions to memorise

Term	Definition	Page
File	A named location on a secondary storage media where data are permanently stored for later access.	19
Text file	A file that consists of human-readable characters (alphabets, numbers, special symbols) stored as ASCII/UNICODE bytes; lines are separated by EOL (`\n`).	20
Binary file	A file that stores data as a stream of bytes representing actual content (images, audio, executables, etc.); not human-readable.	21
File handle (`file_object`)	The object returned by `open()`; establishes a link between the program and the data file on permanent storage.	21
Access mode	An optional argument to `open()` that specifies how the file is to be processed (read, write, append, binary, etc.).	22
EOL (End of Line)	A special character that terminates each line of a text file; default in Python is `\n`.	20
Serialization (Pickling)	The process of transforming a Python object in memory to a byte stream (stored in a binary file or sent over a network). Also called pickling.	32
De-serialization (Unpickling)	The inverse of pickling; converting a byte stream back to a Python object.	32
`seek()`		28

Term	Definition	Page
	A method that repositions the file object to a specified byte offset from a reference point (0 = beginning, 1 = current, 2 = end).	
`tell()`	A method that returns the current byte position of the file object from the beginning of the file.	28
`with` clause	A Python construct for opening files that automatically closes the file when the block exits, even on exceptions.	23
`flush()`	A method that forcefully writes buffer contents to the file without closing it.	24
`open()`	Built-in function that opens a file and returns a file object	21
`close()`	File method that flushes the buffer and frees the file resources	23
`write()`	File method that writes a single string to a text file	24
`writelines()`	File method that writes an iterable of strings	24
`read([n])`	File method that returns n bytes (entire file if no argument)	25
`readline([n])`	File method that reads one line up to n bytes	26
`readlines()`	File method that reads all lines into a list	26
`pickle.dump()`	Pickle function that serializes a Python object into a binary file	32
`pickle.load()`	Pickle function that deserializes a Python object from a binary file	33
`EOFError`	Exception raised by `pickle.load()` when end of file is reached	33
`wb` mode	Binary write mode — required for `pickle.dump()`	32
`rb` mode	Binary read mode — required for `pickle.load()`	33
Buffer	In-memory holding area for data before it is written to disk	24

2.3 Diagrams / processes to remember

- **Table 2.1 — File Open Modes** (p. 22): A tabular summary of all access modes (`r` , `rb` , `r+` , `w` , `wb+` , `a` , `a+`) with their descriptions and initial file offset positions. Students must memorize which modes start at the beginning vs. the end of the file, and which modes overwrite vs. append.
- **Figure 2.1 — Contents of myfile.txt** (p. 25): Shows the output of `writelines()` in Notepad — three lines written without explicit `\n` separators display correctly only because `\n` was embedded in each string. Reinforces that `writelines()` does not auto-insert newlines.
- **Program 2-2 — seek() and tell() demonstration** (p. 28–29): Illustrates the file object moving from byte 33 (after `read()`) back to 0 (after `seek(0)`) and then to byte 10 (after `seek(10)`). Key to understanding random access.

- **Program 2-8 — Binary file with pickle** (p. 34–35): Shows the complete workflow — import pickle → open in `ab` mode → `dump()` records → close → open in `rb` mode → `load()` records in a `try…except EOFError` loop. The output shows each employee record loaded as a list.

2.4 Common confusions / NTA trap points

- `write()` **vs** `writelines()` **:** `write()` accepts only a single string and returns the character count; `writelines()` accepts an iterable of strings and returns `None`. Neither method automatically adds `\n` — the student must include it. NTA often gives code snippets and asks what the output or return value will be.
- **Default file mode and type:** If no access mode is specified in `open()`, Python opens the file in read (`r`) text mode. Many MCQs test whether students know that `w` truncates existing content while `a` preserves it.
- `read()` **with no argument vs** `readline()` **:** `read()` with no argument reads the entire file as one string; `readline()` reads exactly one line including the `\n`. Confusing the two leads to wrong answers in code-trace questions.
- `seek()` **reference point values:** 0 = beginning, 1 = current position, 2 = end. NTA likes distractor options that swap these values or claim the default reference point is 1 instead of 0.
- **Pickle file modes:** `dump()` requires the file to be opened in **binary write** (`wb`) or **binary append** (`ab`) mode; `load()` requires **binary read** (`rb`) mode. Opening a pickle file in text mode (`w` or `r`) is a common trap.
- `with` **clause auto-close (NCERT §2.3.3, p. 23).** Students sometimes think the file remains open after the `with` block. The file is automatically closed when the block exits.
- `w` **mode truncates! (NCERT Table 2.1, p. 22).** Opening an existing file in `w` deletes its contents. Use `a` to preserve and append. NTA exploits this.
- `open()` **without mode defaults to** `r` **text (NCERT §2.3.1, p. 22).** NTA distractor: claims default is `w`.
- **EOF in** `readline()` **returns empty string, not None (NCERT §2.5.2, p. 26).** Use `while line := readline():` style or test for `''`.
- `pickle.dump()` **writes ONE object per call (NCERT §2.8.1, p. 32).** To store multiple, call dump in a loop.
- `tell()` **returns BYTES not lines (NCERT §2.6.1, p. 28).** It is a byte offset.
- `seek()` **reference_point in text mode (NCERT §2.6.2, p. 28).** For text files, NCERT recommends only `seek(offset, 0)` — moving by negative offsets in text mode can be problematic.

🎯 Practice MCQs

Q1. Which of the following is the correct syntax to open a file named `data.txt` for both reading and writing, with the file offset positioned at the beginning of the file?

A. `open("data.txt", "a+")`

B. `open("data.txt", "r+")`

C. `open("data.txt", "rb")`

D. `open("data.txt", "w")`

Q2. Consider the following code: ```python myobject = open("myfile.txt", "w") myobject.write("Hello World\n") myobject.close() ``` Which of the following statements is TRUE about the `write()` call above?

A. It writes the string to the file immediately on disk.

B. It writes data to a buffer; the buffer is moved to permanent storage when `close()` is called.

C. It returns `None` since no value is explicitly returned.

D. It raises a `TypeError` because integers cannot be passed to `write()`.

Q3. What will be the output of the following code, assuming `myfile.txt` contains the three lines: `Hello everyone`, `Writing multiline strings`, `This is the third line`? ```python myobject = open("myfile.txt", "r") print(myobject.readlines()) myobject.close() ```

A. `'Hello everyone Writing multiline strings This is the third line'`

B. `['Hello everyone', 'Writing multiline strings', 'This is the third line']`

C. `['Hello everyone\n', 'Writing multiline strings\n', 'This is the third line']`

D. `('Hello everyone\n', 'Writing multiline strings\n', 'This is the third line')`

PYQ Alignment

File Handling in Python is a consistently tested topic in CUET Computer Science, typically contributing 2–4 direct questions per year, most commonly targeting file open modes (Table 2.1), the distinction between `read()` / `readline()` / `readlines()`, and the pickle module's `dump()` and `load()` methods. Statement-based and code-trace questions on `seek()` / `tell()` and the `with` clause have appeared in recent years as medium-hard items. See PYQ archive for Computer Science.