1. To what does a relative path refer?

   A relative path refers to the path to a file or directory that is relative to the current working directory of the user or the program. In other words, it is the path to a file or directory from the current location in the directory hierarchy.

   For example, if the current working directory is "C:\Users\Username\Documents", and there is a file called "example.txt" in a directory called "Folder1" located at "C:\Users\Username\Documents\Folder1", then the relative path to "example.txt" would be "Folder1\example.txt".

   Relative paths are often used to navigate to files or directories that are located in the same folder or in a subdirectory of the current working directory, and they can be shorter and easier to read than absolute paths, which specify the full path to a file or directory starting from the root directory.

2. What does an absolute path start with your operating system?

   The format of an absolute path in an operating system depends on the file system and the conventions of that operating system. However, in most modern operating systems, an absolute path starts with the root directory of the file system, which is usually represented by a forward slash ("/") character.

   For example, in Unix-like systems such as Linux and macOS, an absolute path might look like "/home/username/documents/example.txt", where the first forward slash represents the root directory, and "home", "username", "documents", and "example.txt" represent directories and files located at specific locations in the file system.

   In Windows, an absolute path might look like "C:\Users\Username\Documents\example.txt", where "C:" represents the root directory of the primary partition of the hard drive, and "Users", "Username", "Documents", and "example.txt" represent directories and files located at specific locations in the file system.

   Note that absolute paths always specify the full path to a file or directory from the root directory, regardless of the current working directory of the user or the program.

3. What do the functions os.getcwd() and os.chdir() do?

   The functions os.getcwd() and os.chdir() are part of the Python os module and are used to get and change the current working directory of a Python program, respectively.

   os.getcwd(): This function returns the current working directory (CWD) of the Python program as a string. The CWD is the directory where the program is currently executing, and it is the default location where the program will look for files if their path is not specified explicitly. For example, if a Python program is executed from the directory "C:\Users\Username\Documents", os.getcwd() would return the string "C:\Users\Username\Documents".

   os.chdir(path): This function changes the current working directory of the Python program to the directory specified by the path parameter. The path parameter can be an absolute or

relative path to a directory. For example, if the current working directory is "C:\Users\Username\Documents", and os.chdir("C:\Users\Username\Downloads") is called, the current working directory will be changed to "C:\Users\Username\Downloads".

It is important to note that changing the current working directory affects the behavior of functions and modules that rely on relative paths, as the current working directory is used as the starting point for these paths. Therefore, care should be taken when using os.chdir() to avoid unexpected behavior in the program.

4. What are the . and .. folders?

In most file systems, including those used in modern operating systems such as Windows, Linux, and macOS, the . and .. folders are special directories that are used to represent the current directory and the parent directory, respectively.

. (dot): This represents the current directory. In other words, it refers to the directory that the user or program is currently in. For example, if the user is currently in the directory "C:\Users\Username\Documents", then "." would represent "C:\Users\Username\Documents".

.. (dot dot): This represents the parent directory. In other words, it refers to the directory that is one level up in the directory hierarchy from the current directory. For example, if the user is currently in the directory "C:\Users\Username\Documents", then ".." would represent "C:\Users\Username".

The . and .. folders are used frequently in navigating the file system using relative paths. For example, if the user is in the directory "C:\Users\Username\Documents\Folder1", and there is a file called "example.txt" in the directory "C:\Users\Username\Documents\Folder2", then the relative path to "example.txt" from "Folder1" would be "..\Folder2\example.txt", where ".." refers to the parent directory "Documents".

It is important to note that the . and .. folders are not the only special directories in file systems, and there may be other directories with special meanings depending on the file system and operating system being used.

5. In C:\bacon\eggs\spam.txt, which part is the dir name, and which part is the base name?

In the file path "C:\bacon\eggs\spam.txt":

"C:" is the drive name (or root directory) of the file system.

"bacon\eggs" is the directory name, which is the path of the directory containing the file "spam.txt".

"spam.txt" is the base name of the file, which is the name of the file itself with the file extension.

Therefore, the directory name is "bacon\eggs", and the base name is "spam.txt".

6. What are the three “mode” arguments that can be passed to the open() function?

The open() function in Python is used to open a file and returns a file object. It takes two arguments: the path to the file and the mode in which the file should be opened. The mode

argument specifies the purpose for which the file is opened and what operations are allowed on it. There are three main mode arguments that can be passed to the open() function:

r (read mode): This mode is used to open a file for reading its contents. If the file does not exist, it will raise a FileNotFoundError exception.

w (write mode): This mode is used to open a file for writing its contents. If the file does not exist, it will be created. If the file already exists, its contents will be truncated (erased) before writing new data.

a (append mode): This mode is used to open a file for appending new data to its existing contents. If the file does not exist, it will be created.

These mode arguments can be used alone or in combination with other modes to specify the exact behavior of the file object. For example, adding a + to a mode (e.g., r+, w+, a+) will allow both reading and writing operations on the file. Adding a b to a mode (e.g., rb, wb, ab) will open the file in binary mode instead of text mode.

7. What happens if an existing file is opened in write mode?

If an existing file is opened in write mode using the w argument with the open() function in Python, the file's contents will be truncated, and any existing data in the file will be erased. In other words, the file will be completely overwritten with new data.

If the file does not already exist, it will be created when it is opened in write mode.

Therefore, opening a file in write mode can be a destructive operation, as it erases any existing data in the file. It is essential to use write mode with caution and make sure that me intend to overwrite the existing data in the file. If we want to append new data to an existing file without erasing its contents, we should use append mode (a) instead of write mode (w).

8. How do you tell the difference between read() and readlines()?

In Python, read() and readlines() are both methods used to read data from a file, but they work in different ways.

The read() method reads the entire contents of a file and returns it as a single string. It reads from the current position of the file pointer until the end of the file is reached or until the specified number of bytes have been read.

The readlines() method reads the entire contents of a file and returns it as a list of strings, where each string represents a single line of the file. It reads from the current position of the file pointer until the end of the file is reached.

9. What data structure does a shelf value resemble?

In Python, a shelf is a persistent dictionary-like object that is implemented as a database file. It is similar to a dictionary in that it stores key-value pairs, but it differs in that it can be accessed and modified like a dictionary, but its contents are stored on disk instead of in memory.

The shelf module uses the pickle module to store Python objects as binary data in the database file, which allows complex data structures to be stored in the shelf. When we access a value in a shelf, the shelf will read the corresponding data from the disk and deserialize it back into a Python object.

Since a shelf stores key-value pairs like a dictionary, it can be seen as a dictionary-like data structure. However, unlike a dictionary, a shelf's contents are stored on disk, which allows it to store much larger amounts of data than would be possible to store in memory. Therefore, a shelf can be thought of as a persistent dictionary that can be used to store large amounts of data that need to be accessed frequently.

In [ ]: