**1. To what does a relative path refer?**

Ans. A relative path refers to the location of a file or directory relative to the current working directory or another specified reference point. It is a path that is expressed in relation to another location rather than being an absolute path that specifies the complete path from the root directory.

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

Ans. In most operating systems, including Windows, macOS, and Linux/Unix-based systems, an absolute path starts with the root directory. The root directory is the highest-level directory in the file system hierarchy and serves as the starting point for navigating the file system.

In different operating systems, the representation of the root directory may vary:

**For Windows:** An absolute path starts with a drive letter (such as C:, D:, etc.) followed by a backslash (). For example: "C:\path\to\file".

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

Ans. The functions os.getcwd() and os.chdir() are part of the os module in Python and are used for working with the current working directory.

**os.getcwd():** The os.getcwd() function returns a string representing the current working directory (CWD) of the Python script or the current process. The current working directory is the directory in which the script is running or the directory from which the Python interpreter was launched. It provides information about the current location in the file system.Here's an example of how to use os.getcwd():

***import os current\_directory = os.getcwd() print(current\_directory)***

This will output the current working directory.

**os.chdir(path**): The os.chdir(path) function is used to change the current working directory to the specified path. It takes a string argument path representing the directory path to which you want to change the working directory.Here's an example of how to use os.chdir():

***import os os.chdir('/path/to/directory')***

By using os.chdir(), you can navigate and switch between different directories within the file system hierarchy, allowing your script or program to operate in a specific directory context.

**4. What are the . and .. folders?**

Ans. **. (dot):** The . (dot) folder represents the current directory. It is used as a reference to the current location in the file system. When you specify a file path with a relative path and start with ., it indicates that the file or directory is located in the current directory. For example, if you are in the directory /home/user and you refer to a file as ./file.txt, it means the file is located in the current directory /home/user.

**.. (dot dot):** The .. (dot dot) folder represents the parent directory. It is used to refer to the directory immediately above the current directory in the file system hierarchy. When you specify a file path with a relative path and start with .., it indicates that the file or directory is located in the parent directory. For example, if you are in the directory /home/user/documents and you refer to a file as ../file.txt, it means the file is located in the parent directory /home/user.The **.** and **..** folders are used for navigation and referencing within the file system. They provide a way to specify relative paths and move between directories without explicitly mentioning the directory names. They are commonly used in command-line interfaces and file system operations to refer to the current directory and its parent directory, respectively.

It's important to note that . and .. are special folders and do not represent actual files or directories on the file system. They are conventions used to provide a consistent and intuitive way to reference locations in the file system hierarchy.

Top of Form

Bottom of Form

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

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

The directory name (dir name) is "C:\bacon\eggs". It represents the path to the directory where the file "spam.txt" is located. The directory name includes all the directories leading up to the file.

The base name is "spam.txt". It represents the actual name of the file itself, without the directory path. It is the name by which the file is identified within its containing directory.

To clarify further, in the file path "C:\bacon\eggs\spam.txt":

"C:\bacon\eggs" is the directory name.

"spam.txt" is the base name.

Understanding the distinction between the directory name and the base name is useful when working with file paths and performing operations such as navigating directories, creating new files, or manipulating file names.

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

Ans. The open() function in Python accepts three different "mode" arguments as the second parameter, specifying the purpose or intention of opening a file. These modes determine how the file will be opened and what operations can be performed on it. The three common mode arguments are:

"r" - Read mode: This mode is used for opening a file for reading. The file must exist in this mode, otherwise, an error will occur. When a file is opened in read mode, you can only perform read operations on it, such as reading its contents or iterating over its lines. Attempting to write or modify the file will result in an error.

"w" - Write mode: This mode is used for opening a file for writing. If the file already exists, its previous contents will be truncated (erased) upon opening. If the file doesn't exist, a new file will be created. In write mode, you can perform write operations to the file, such as writing new data or overwriting its contents. Note that opening a file in write mode will discard the previous data in the file.

"a" - Append mode: This mode is used for opening a file in append mode, where new data is added at the end of the file. If the file exists, the new data will be appended to its existing contents. If the file doesn't exist, a new file will be created. In append mode, you can only perform write operations by adding data to the end of the file, and you cannot modify or overwrite existing data in the file.

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

Ans. When an existing file is opened in write mode ("w"), the previous contents of the file are completely truncated (erased) upon opening. In other words, the file is emptied before any new data is written to it.Here are the key points to understand about opening an existing file in write mode:

If the file exists and is successfully opened in write mode, its previous contents are completely removed, and the file becomes empty.

If the file does not exist at the specified path, a new file will be created with the given name.

Any subsequent write operations performed on the file in write mode will start writing data from the beginning of the file.

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

Ans. **read():** The read() method reads the entire contents of a file as a single string. It reads from the current file position up to the end of the file or the specified number of characters if an argument is provided. The returned string includes newline characters ('\n') to represent line breaks in the file.

Example usage:

**with open("file.txt", "r") as file:**

**content = file.read()**

In this example, file.read() will return the entire contents of the file as a single string, including newline characters if present.

**readlines():** The readlines() method reads the contents of a file line by line and returns a list of strings, where each element represents a line from the file. The newline characters ('\n') at the end of each line are included in the strings.

Example usage:

**with open("file.txt", "r") as file:**

**lines = file.readlines()**

In this example, file.readlines() will return a list of strings, where each string represents a line from the file.

**9. What data structure does a shelf value resemble?**

Ans. In Python, a shelf value resembles a dictionary-like data structure. The shelf value is provided by the shelve module, which allows you to store and retrieve Python objects persistently using a key-value storage system.The shelf value can be seen as a persistent dictionary, where the keys and values are stored on disk. It provides a simple interface similar to a dictionary for storing, retrieving, and modifying data. The keys in a shelf are unique, and each key is associated with a corresponding value.