1.To what does a relative path refer?

**Ans.** A relative path refers to a file or directory location that is relative to the current location or the location of the file that you are working with. In other words, a relative path provides a path to a file or directory based on the current directory or the location of the reference file, rather than an absolute path that provides a full path starting from the root directory.

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

**Ans.** An absolute path in most operating systems starts with the root directory represented by a forward slash "/".

For example, in Unix-based systems such as Linux and macOS, an absolute path might look like "/user/local/bin".

In Windows, an absolute path might look like "C:\Windows\System32".

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

**Ans.** `os.getcwd()` is a function from the `os` module in Python, which stands for "operating system". The function `os.getcwd()` stands for "get current working directory". When you call this function, it returns a string representing the current working directory of your Python process. The current working directory is the directory from which your Python script was run and it is used as the default location for relative file paths.

`os.chdir()`, on the other hand, stands for "change directory". The function `os.chdir()` allows you to change the current working directory of your Python process. You can use this function to change the current working directory to any directory on your file system. When you change the current working directory using `os.chdir()`, any subsequent file and directory operations will be relative to the new working directory.

Here's a simple example of how you might use these functions:

import os

print("Original working directory:", os.getcwd())

# Change the current working directory

os.chdir("/tmp")

print("New working directory:", os.getcwd())

4. What are the . and .. folders?

**Ans**. The ". and .." folders refer to two special directories in a file system. They are present in every directory on a Unix-based operating system, including Linux and macOS.

". (dot)" represents the current directory. It's a shortcut to refer to the directory in which it is located. For example, if you are in the "/home/user/documents" directory, then "./documents" is a reference to the same directory.

".. (dot dot)" represents the parent directory. It's a shortcut to refer to the directory that is one level higher in the file system hierarchy. For example, if you are in the "/home/user/documents" directory, then "../documents" is a reference to the "/home/user" directory.

These special directories are used to navigate the file system, and they are often used in scripts and other programming tasks.

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", "C:\bacon\eggs" is the directory name, and "spam.txt" is the base name. The directory name refers to the path specifying the location of the file, while the base name refers to the actual name of the file.

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

**Ans.** The three "mode" arguments that can be passed to the `open()` function in Python are:

**"r" (read mode) -** opens the file for reading only. This is the default mode, and it raises an error if the file does not exist or if you don't have sufficient permissions to access the file.

**"w" (write mode) -** opens the file for writing. If the file exists, its contents will be overwritten. If the file does not exist, a new file with the same name will be created.

**"a" (append mode) -** opens the file for writing and appends new data to the end of the file. If the file does not exist, a new file with the same name will be created. Unlike "w" mode, this mode will not overwrite the existing contents of the file.

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

**Ans.** If a file is opened in write mode and it already exists, the contents of the file will be deleted and a new file will be created. This means that anything that was previously in the file will be lost and replaced with new data that is written to the file. In other words, opening an existing file in write mode effectively truncates the file, making it empty and ready to be written to again. It's important to use caution when opening files in write mode, as the contents of the file may be permanently lost if you are not careful.

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

**Ans.** In Python, `read()` and `readlines()` are both methods that can be used to read the contents of a file. The difference between them lies in how they return the data.

`read()` method reads the entire file and returns its contents as a single string. This method is useful when you want to read the entire file into memory, for example when you want to process the contents of the file line by line.

`readlines()` method reads the entire file and returns its contents as a list of strings, where each string is a line from the file. This method is useful when you want to access the contents of the file line by line, for example when you want to search for a specific line or pattern in the file.

So, the main difference between `read()` and `readlines()` is how they return the data. If you want to process the entire file as a single string, you can use `read()`. If you want to process the file line by line, you can use `readlines()`.

9. What data structure does a shelf value resemble?

**Ans.** A shelf value resembles a dictionary data structure.