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

**Answer:**

A relative path refers to the location of a file or directory relative to the current working directory or another reference point.

In the context of file systems, a relative path specifies the path to a file or directory starting from the current directory or a known location. It does not provide the complete path from the root of the file system. Instead, it describes the path in relation to the current working directory.

For example, let's assume the current working directory **is /home/user/** and there is a file named example.txt located in the directory **/home/user/documents/.** The relative path to example.txt from the current working directory would be documents/example.txt.

Relative paths can also include references to parent directories using special notations. For example, ../ refers to the parent directory, allowing you to navigate up the directory hierarchy. So, if the current working directory is **/home/user/documents/,** and you want to reference a file in the parent directory, the relative path could be **../parent\_file.txt.**

It's important to note that the interpretation of a relative path depends on the current working directory or the reference point from which it is resolved. Changing the current working directory or the reference point can affect the outcome of a relative path.

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

**Answer:**

**Windows:** In Windows, an absolute path typically starts with a drive letter followed by a colon (:) and uses backslashes (\) to separate directory levels.

**For example:**

**C:\Users\Username\Documents\file.txt**

**Unix-like systems (Linux, macOS, etc.):** In Unix-like systems, an absolute path starts with a forward slash (/) and uses forward slashes as directory separators.

**For example:**

**/home/username/Documents/file.txt**

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

**Answer:**

**os.getcwd():** This function returns a string representing the current working directory. The working directory is the directory in which the Python script is currently executing. It can be the directory from which the script was launched or the directory specified when running the script

**os.chdir(path):** This function changes the current working directory to the specified path. The path parameter is a string representing the desired directory to which you want to navigate. It can be an absolute path or a relative path. After calling os.chdir(), the current working directory will be updated to the specified path.

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

**Answer:**

**. (Dot**): The dot represents the current directory. It is used to refer to the directory you are currently in. When specifying a file path, using . means referring to the current directory.

.. (Double Dot): **The double dot represents the parent directory. It is used to refer to the directory** one level above the current directory. When specifying a file path, using .. means referring to the parent directory.

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

**Answer:**

In the path **C:\bacon\eggs\spam.txt**, the directory name and the base name can be identified as follows:

Directory Name: **C:\bacon\eggs**

The directory name refers to the path of the directory that contains the file. In this case, the directory name is C:\bacon\eggs. It represents the location of the file spam.txt.

**Base Name: spam.txt**

The base name refers to the actual file name without the directory path. In this case, the base name is spam.txt.

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

**Answer:**

**Read Mode ('r'):** This is the default mode and allows you to open a file for reading. It is used when you want to access the contents of an existing file. In read mode, the file pointer is positioned at the beginning of the file, and if the file does not exist, an error will occur.

**Write Mode ('w'):** This mode is used when you want to create a new file or overwrite the contents of an existing file. If the file does not exist, it will be created. If it does exist, the existing contents will be truncated and replaced with the new data. In write mode, the file pointer is positioned at the beginning of the file.

**Append Mode ('a'):** This mode is used when you want to append new data to the end of an existing file without overwriting the existing contents. If the file does not exist, it will be created. In append mode, the file pointer is positioned at the end of the file.

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

**Answer:**

If an existing file is opened in write mode ('w') using the open() function in Python

If the file already exists, its contents will be completely overwritten.

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

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

**Answer:**

**read():** The read() method is used to read the entire contents of a file as a single string. It reads and returns the entire file's content, including newline characters ('\n'), as a single string. If you do not specify a parameter, it will read the entire file. If you provide an optional parameter size, it will read and return only the specified number of characters.

**file = open('example.txt', 'r')**

**content = file.read() # Reads the entire file as a string**

**print(content)**

**file.close()**

**readlines():** The readlines() method is used to read the contents of a file line by line and returns them as a list of strings. Each line of the file becomes an element in the returned list. It includes the newline character ('\n') at the end of each line, which can be removed using string methods if desired.

**file = open('example.txt', 'r')**

**lines = file.readlines() # Reads the file line by line as a list of strings**

**for line in lines:**

**print(line)**

**file.close()**

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

**Answer:**

In Python, a shelf value, as obtained from the shelve module, resembles a dictionary-like data structure.

The shelve module provides a simple persistent storage option for Python objects by utilizing a dictionary-like interface. It allows you to store and retrieve data using key-value pairs. Under the hood, the shelve module uses a database-like file to store the data.

**import shelve**

**Open a shelf file**

**with shelve.open('mydata') as shelf:**

**# Store values using keys**

**shelf['name'] = 'John'**

**shelf['age'] = 30**

**# Retrieve values using keys**

**name = shelf['name']**

**age = shelf['age']**

**print("Name:", name)**

**print("Age:", age)**

**# Iterate over keys and values**

**for key in shelf:**

**value = shelf[key]**

**print(key, "=", value)**