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** of the user or application.

In other words, a relative path specifies the path to a file or directory starting from the current working directory, **rather than from the root directory** of the file system **as in the case of an absolute path**.

For **example**, suppose the **current working directory** is **"/home/user/documents/"** and you want to refer to a **file** named "**example.txt**" located **in** a **subdirectory** named "**project**" **within** the **current** **working directory**. The **relative path** to that file would be **"./project/example.txt"**, where the **dot** **represents** the **current working directory**.

**Similarly**, if you want to refer to a file named "**example.txt**" **located** **in a parent directory of** the **current working directory**, you can use the relative path **"../example.txt"**, where the **double dots represent the parent directory**.

In both Windows and Mac/Linux operating systems, a **relative path** **specifies** the **path to a file or directory relative to the current working directory** of the user or application.

In **Windows**, the **path separator** is a **backslash "\".** For **example**, **to refer** to a file named "**example.txt**" located **in a subdirectory named "project**" **within** the **current working directory**, the **relative path** would be **".\project\example.txt**". **Similarly**, **to refer** to a **file** **in a parent directory**, you can use the **relative path** **"..\example.txt".**

In **Mac/Linux**, the **path separator** is a **forward slash "/".** For example, to refer to a file named "**example.txt**" located **in a subdirectory named "project**" **within** the **current working directory**, the **relative path** would be **"./project/example.txt".** **Similarly**, **to refer** to a **file** **in a parent directory**, you can use the **relative path "../example.txt".**

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

Ans: In **most operating systems**, including Windows, macOS, and Linux, an **absolute path starts with** the **root directory** of the file system.

In **Windows**, the **root directory** is typically represented by the **drive letter** **followed by a colon**, such as "**C:**". For example, an absolute path to a file "example.txt" located in the root directory of the C drive would be "C:\example.txt".

In **macOS** and **Linux**, the **root directory** is represented by a **forward slash ("/").** For example, an absolute path to a file "example.txt" located in the root directory of a Linux or macOS system would be "/example.txt".

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

Ans: **os.getcwd()** returns the current working directory (the directory where the Python script is currently running).

**os.chdir(path)** changes the current working directory to the path specified.

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

Ans: **` .` refers** to the **current directory** and **`..`** **refers** to the **parent directory of the current directory**. These are special folders used in file systems to navigate and reference directories relative to the current location. The `.` folder is used to indicate the current directory when specifying file paths, while `..` is used to navigate to the parent directory.

1. **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:"** is the **drive name** (the **root directory** of the C drive)
* **"bacon\eggs"** is the **directory name** (the folder that contains the file)
* **"spam.txt"** is the **base name** (the **actual name** of the file)

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

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

Ans: The **open() function** in Python can accept the **following modes**:

* "**r**" (**read mode**): This is the **default mode** and opens the file for reading. Raises an error if the file doesn't exist.
* "**w**" (**write mode**): This mode opens the file for **writing**. If the **file doesn't exist**, it **creates** a new file. **If** the file already **exists**, it **overwrites** the existing content.
* "**a**" (append mode): This mode opens the file for **writing** and **appends** any new data written **to the end of the file**. **If** the **file doesn't exist**, it **creates** a new file.

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

**Ans:** "**w**" (**write mode**): This mode opens the file for **writing**. If the **file doesn't exist**, it **creates** a new file. **If** the file already **exists**, it **overwrites** the existing content.

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

Ans: In Python, **read()** and **readlines()** are both **file methods** used **to read data** **from** **a file object**. However, they **differ** **in how they read** the data:

**read()** **reads** the **entire content** of the file **as a single string**, **including newline characters ("\n")**, and **returns** it as **a string object**.

**readlines() reads** the **entire content** of the file **line-by-line,** and **returns** a **list** **of strings**, where each string represents a line from the file. By default, **each line includes** the **newline character at the end ("\n").**

In summary, read() returns a single string object that contains the entire file content, while readlines() returns a list of strings, where each string represents a line from the file.

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

Ans: **In Python**, a **shelf** value **resembles** a **dictionary data structure**. It is **a built-in module in Python** that **provides** a **way to store** and **persistently** **manipulate** **data** **in a dictionary-like object**. A shelf is a persistent, dictionary-like object that is **stored in a file on disk**, and is **accessed using** the **same syntax as** for **dictionaries**.

Similar to dictionaries, shelves **have keys and values**, and allow fast lookup of values based on their keys. **However**, **unlike dictionaries**, shelves **store** **their** **data persistently on disk**, which **means** that the **data remains available even after the program exits**. This makes shelves **useful** for **storing and retrieving** large amounts of **data that needs to be persisted across multiple program runs**.