1. To what does a relative path refer?

A relative path refers to the location of a file or directory in relation to the current working directory. It specifies the path to a file or directory starting from the current location, rather than starting from the root directory.

In a file system, the current working directory is the directory from which a program or script is executed or the directory you are currently working in when navigating through the file system. A relative path is defined with respect to this current working directory.

Relative paths are typically used when referring to files or directories within the same project or directory structure. They provide a way to specify the location of a file or directory relative to the current location without specifying the full absolute path from the root directory.

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

The absolute path in an operating system starts with the root directory.

The root directory is the topmost directory in a file system hierarchy. It serves as the starting point for navigating the entire file system. The notation for the root directory varies depending on the operating system:

* In Unix-like systems (such as Linux and macOS), the absolute path starts with a forward slash (/). For example: /home/user/documents/file.txt.
* In Windows systems, the absolute path starts with a drive letter followed by a colon (:) and a backslash (\). For example: C:\Users\user\Documents\file.txt.

In both cases, the absolute path provides a complete and specific location of a file or directory within the file system. It includes all the necessary information to navigate directly to that file or directory, starting from the root directory.

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

os.getcwd():

* os.getcwd() returns a string representing the current working directory.
* The current working directory is the directory from which a Python script or program is being executed or the directory you are currently working in when interacting with the file system.

os.chdir(path):

* os.chdir(path) changes the current working directory to the specified path.
* The path argument is a string that represents the path to the directory you want to set as the new current working directory.

1. What are the . and .. folders?

The . and .. folders are special directory references commonly found in file systems. Here's their significance:

1. . (dot):
   * The . folder represents the current directory or the current location in the file system.
   * It is used to refer to the directory itself when specifying paths or performing operations within the current directory.
   * For example, if you are in the /home/user directory, . refers to /home/user itself.
2. .. (dot dot):
   * The .. folder represents the parent directory or the directory that contains the current directory.
   * It is used to refer to the parent directory when specifying paths or navigating to a directory higher up in the hierarchy.
   * For example, if you are in the /home/user directory, .. refers to the /home directory.
3. In C:\bacon\eggs\spam.txt, which part is the dir name, and which part is the base name?

In the path C:\bacon\eggs\spam.txt, the directory name (dir name) refers to the part that represents the directory containing the file, and the base name refers to the actual file name itself.

example:

* Directory name (dir name): C:\bacon\eggs
  + This represents the directory that contains the file spam.txt.
  + It includes the full path to the directory, starting from the root directory (C:) and then specifying the subsequent directories (bacon and eggs).
* Base name: spam.txt
  + This is the name of the file itself, without including the path or any directory information.

So, in C:\bacon\eggs\spam.txt:

* Directory name (dir name): C:\bacon\eggs
* Base name: spam.txt

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

The open() function in Python can take several mode arguments to specify how a file should be opened. The three commonly used mode arguments are:

1. "r" (Read mode):
   * This is the default mode when no mode argument is provided.
   * It opens the file for reading.
   * If the file does not exist, it raises a FileNotFoundError.
   * Example: open("file.txt", "r")
2. "w" (Write mode):
   * It opens the file for writing.
   * If the file does not exist, it creates a new file. If the file exists, it truncates it (empties its contents) before writing.
   * Example: open("file.txt", "w")
3. "a" (Append mode):
   * It opens the file for appending data.
   * If the file does not exist, it creates a new file. If the file exists, it appends the new data to the existing contents.
   * Example: open("file.txt", "a")
4. What happens if an existing file is opened in write mode?

If an existing file is opened in write mode ("w"), the file is truncated (emptied) before any new data is written into it. In other words, the previous contents of the file are completely removed.

Here's what happens when an existing file is opened in write mode:

1. If the file exists:
   * All existing content within the file is deleted.
   * The file is effectively "reset" to an empty state.
   * The file is ready to be written with new data.
2. If the file does not exist:
   * A new file with the specified name is created.
   * It is an empty file ready to be written with new data.
3. How do you tell the difference between read() and readlines()?

read():

* The read() method reads the entire contents of a file as a single string.
* It reads the file from the current position until the end of the file, or until a specified number of characters (size) if provided.
* If no size argument is specified, it reads the entire file.
* The returned string includes newline characters (\n) and preserves the original formatting of the file.

readlines():

* The readlines() method reads the entire contents of a file and returns a list of strings.
* It reads the file from the current position until the end of the file, or until a specified number of bytes (size) if provided.
* Each line in the file is stored as a separate string element in the list.
* The newline characters (\n) are also included in each string element.

1. What data structure does a shelf value resemble?

A shelf value in Python resembles a dictionary-like data structure.

In Python, a shelf is a persistent, dictionary-like object that acts as a persistent storage mechanism for Python objects. It is provided by the shelve module, which allows you to store and retrieve objects using key-value pairs.

A shelf value behaves similarly to a dictionary in the sense that it supports operations like accessing values by keys, adding or updating values, and iterating over the key-value pairs. However, unlike dictionaries, shelf values persist on the disk even after the program terminates, making them useful for saving and loading data between different program executions.