**Assignment 9**

Q1. To what does a relative path refer?

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

Q2. What does an absolute path start with your operating system?

Sol. In most operating systems, an absolute path starts with the root directory. The root directory is the top-level directory in the file system hierarchy and serves as the starting point for all other directories and files.

The specific representation of the root directory may vary depending on the operating system:

On Windows systems, the root directory is typically represented by the drive letter followed by a colon (**:**), such as **C:\** for the C drive or **D:\** for the D drive.

On Unix-like systems (including Linux and macOS), the root directory is denoted by a forward slash (**/**).

Q3.

Sol. In Python, the functions **os.getcwd()** and **os.chdir()** are part of the **os** module and are used to interact with the file system.

**os.getcwd()**:

os.getcwd() returns the current working directory (CWD) as a string.

The current working directory is the directory in which your Python script is currently running.

When you execute os.getcwd(), it will provide the absolute path of the current working directory.

os.chdir(path):

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

The path argument should be a string representing a valid directory path.

After calling os.chdir(path), the current working directory will be updated to the specified directory.

Q4. What are the . and .. folders?

Sol. In file systems, the **.** and **..** folders (or directories) have special meanings:

**.** (dot):

The **.** (dot) folder refers to the current directory.

It is used to represent the current working directory or the location where the command or script is being executed.

For example, if you are currently in the **/home/username/Documents** directory, the **.** folder would refer to **/home/username/Documents**.

**..** (dot-dot):

The **..** (dot-dot) folder refers to the parent directory.

It is used to navigate to the directory immediately above the current directory in the file system hierarchy.

For example, if you are currently in the **/home/username/Documents** directory, the **..** folder would refer to the **/home/username** directory.

The **.** and **..** folders are commonly used in command-line interfaces and file systems to navigate and reference directories. They provide a way to specify relative paths and move between different levels of directory hierarchy.

For instance, if you want to access a file in the parent directory, you can use **../filename** to refer to it. Similarly, if you want to access a file in the current directory, you can simply use **./filename** or omit the **./** part, as it is implied by default.

Q5. In C:\bacon\eggs\spam.txt, which part is the dir name, and which part is the base name?

Sol. In the file path "C:\bacon\eggs\spam.txt", the directory name refers to "C:\bacon\eggs" and the base name refers to "spam.txt".

The directory name (dir name) is the part of the path that specifies the location or folder where the file is stored. In this case, "C:\bacon\eggs" represents the directory name.

The base name is the actual name of the file itself, without the directory path. In this example, "spam.txt" is the base name.

Q6. What are the three “mode” arguments that can be passed to the open() function?

Sol. The "open()" function in Python can accept three different mode arguments:

"r" - This mode is used for reading files. It allows you to open an existing file and read its contents. If the file does not exist, it will result in a FileNotFoundError.

"w" - This mode is used for writing files. It allows you to create a new file or overwrite an existing file. If the file does not exist, a new file will be created. If it already exists, its contents will be truncated (deleted) before writing.

"a" - This mode is used for appending files. It allows you to open an existing file and append new content to the end of the file. If the file does not exist, a new file will be created. The existing contents of the file remain intact.

These modes can be combined with additional characters to modify their behavior. For example, "r+" can be used to open a file for both reading and writing.

Q7. What happens if an existing file is opened in write mode?

Sol. If an existing file is opened in write mode ("w") using the "open()" function in Python, the following will happen:

If the file already exists, its contents will be completely overwritten. The existing data in the file will be deleted, and the file will be empty before any new content is written.

If the file does not exist, a new file will be created with the specified name, and you can start writing content into it.

Therefore, it is essential to exercise caution when using write mode, as it can result in the loss of existing data in the file.

Q8. How do you tell the difference between read() and readlines()?

Sol. In Python, there are two methods for reading data from a file: "read()" and "readlines()". Here's how you can differentiate between them:

1. read(): The "read()" method reads the entire contents of a file as a single string. It reads from the current position in the file until the end and returns the complete content. The resulting string may include newline characters ("\n") and other formatting present in the file. You can use the "read(size)" variant to specify the number of characters to read from the file.

Example usage:

file = open("example.txt", "r")

content = file.read()

print(content)

file.close()

1. readlines(): The "readlines()" method reads the contents of a file line by line and returns a list where each line is an element of the list. It reads from the current position in the file until the end and splits the content into separate lines. The resulting list does not include newline characters ("\n"), as each line is stored as an individual string element.

Example usage:

file = open("example.txt", "r")

lines = file.readlines()

for line in lines:

print(line)

file.close()

Q9. What data structure does a shelf value resemble?

Sol. In Python, the "shelf" module provides a persistent dictionary-like data structure. The data structure that a shelf value resembles is similar to a dictionary. It allows you to store and retrieve key-value pairs, just like a dictionary.

The difference is that a shelf is designed to be persistent, meaning it can be stored on disk and accessed across different program runs. It uses a disk file to store the data, which allows for long-term storage of the dictionary-like object. The shelf data structure is provided by the "shelve" module in Python.