**Python Assignment 9**

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 from the root directory of the file system.

In other words, a relative path provides directions on how to navigate from the current location to the desired file or directory. It is based on the assumption that you have a starting point (the current working directory) and you want to specify a path that is relative to that starting point.

Relative paths often use special symbols to navigate through directories:

. (dot): Refers to the current directory.

.. (double dot): Refers to the parent directory.

For example:

If the current working directory is /home/user/documents, and you want to access a file named report.txt in that directory, the relative path would be simply report.txt.

If you are in the directory /home/user/documents, and you want to access a file named note.txt located in a subdirectory named notes, the relative path would be notes/note.txt.

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

An absolute path starts with the root directory of your operating system. The root directory is the top-level directory in a file system hierarchy and serves as the starting point for specifying the complete path to a file or directory.

The format of an absolute path depends on the operating system:

On Unix-like systems (e.g., Linux, macOS):

An absolute path starts with a forward slash (/) to indicate the root directory. For example: /home/user/documents/report.txt.

On Windows:

An absolute path starts with a drive letter (followed by a colon) and then a backslash (\) to indicate the root directory of that drive. For example: C:\Users\User\Documents\report.txt.

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

The os.getcwd() and os.chdir() functions are part of the os module in Python and are used to work with the current working directory of a Python script or program.

os.getcwd():

The os.getcwd() function returns a string representing the current working directory (CWD) of the Python script or program.

The current working directory is the directory from which the script is being executed.

This function is useful for obtaining the path of the directory where your script is located.

os.chdir(path):

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

This function is used to navigate and change the current working directory to a different directory in the file system.

The path parameter should be a valid directory path.

4. What are the . and .. folders?

In file systems, the . and .. folders (directories) have special meanings and are used to represent specific directory locations:

. (Dot):

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

It is used to specify the current directory in file paths.

For example, if you're in the directory /home/user/documents, the path ./file.txt would refer to a file named file.txt in the current directory.

.. (Double Dot):

The .. (double dot) folder refers to the parent directory.

It is used to navigate to the parent directory in file paths.

For example, if you're in the directory /home/user/documents, the path ../notes/note.txt would refer to a file named note.txt in the notes subdirectory of the parent directory.

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

Dir Name: C:\bacon\eggs

Base Name: spam.txt

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

The open() function in Python takes a "mode" argument that specifies how the file should be opened. The mode argument determines whether the file should be opened for reading, writing, or both, as well as whether the file should be created if it doesn't exist. The three most common mode arguments are:

'r' (Read Mode):

Opens the file for reading (default mode if not specified).

The file pointer is positioned at the beginning of the file.

An error is raised if the file does not exist.

'w' (Write Mode):

Opens the file for writing.

If the file already exists, its contents are truncated (erased).

If the file does not exist, a new file is created.

The file pointer is positioned at the beginning of the file.

'a' (Append Mode):

Opens the file for writing, but does not truncate the file.

If the file already exists, new data is appended to the end of the file.

If the file does not exist, a new file is created.

The file pointer is positioned at the end of the file.

# Read mode

with open('myfile.txt', 'r') as file:

content = file.read()

print(content)

# Write mode

with open('newfile.txt', 'w') as file:

file.write('Hello, world!')

# Append mode

with open('logfile.txt', 'a') as file:

file.write('Log entry: Something happened\n')

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

If an existing file is opened in write mode ('w'), the following things will happen:

File Contents Truncated:

The contents of the existing file will be completely truncated, effectively erasing all the data in the file.

The file will become empty after being opened in write mode.

File Pointer at Beginning:

The file pointer (cursor) is positioned at the beginning of the file.

Any data written using the write() method will overwrite the existing content starting from the beginning of the file.

File Created If Not Exist:

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

It's important to exercise caution when opening a file in write mode, especially if you intend to modify an existing file. Any data present in the file will be permanently lost when it is opened in write mode.

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

Both read() and readlines() are methods that can be used to read data from a file in Python. However, they have different behaviors and return different types of data:

read() Method:

The read() method is used to read the entire content of the file as a single string.

It reads all characters from the current file pointer position (or from the beginning if not specified) up to the end of the file. The returned data is a single string that contains the entire file content.

readlines() Method:

The readlines() method is used to read lines from the file and return them as a list of strings.

It reads all lines from the current file pointer position (or from the beginning if not specified) up to the end of the file.

Each line of text is stored as a separate string element in the returned list.

read() returns a single string containing the entire content of the file.

readlines() returns a list of strings, where each string represents a line from the file.

9. What data structure does a shelf value resemble?

In Python, the shelf module provides a way to work with persistent storage using a dictionary-like interface. The data structure that a shelf value resembles is a dictionary. A shelf is essentially a dictionary that is stored as a file on the disk, allowing you to store and retrieve key-value pairs in a persistent manner.

The shelf module provides the shelve class, which allows you to create and interact with persistent storage files that behave like dictionaries. You can use the shelve class to store and retrieve data, similar to how you would with a dictionary.