1. To what does a relative path refer?

Ans - In Python, a relative path refers to the location of a file or directory relative to the current working directory. The current working directory is the directory from which the Python script or application is being executed.

For example, if your current working directory is "/home/user/" and you have a file named "example.txt" in the "documents" directory, the relative path to "example.txt" would be "documents/example.txt". This is because it is the path relative to the current working directory ("/home/user/").

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

Ans - An absolute path specifies the complete directory hierarchy from the root directory, while a relative path specifies the location of a file or directory with respect to the current working directory.

an absolute path for the file would be something like "/home/user/documents/example.txt," specifying the complete path from the root directory.

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

Ans - In Python, the os.getcwd() function is used to get the current working directory (CWD), and os.chdir(path) is used to change the current working directory.

import os

# Get the current working directory

current\_directory = os.getcwd()

print("Current working directory:", current\_directory)

# Change the current working directory

new\_directory = "/path/to/new/directory"

os.chdir(new\_directory)

# Verify the change

updated\_directory = os.getcwd()

print("Updated working directory:", updated\_directory)

This example first prints the current working directory, then changes the directory using os.chdir(), and finally prints the updated working directory.

1. What are the . and .. folders?

Ans - In Python, the . and .. folders are special notations representing directories in a file system.

. (Dot): The dot (.) refers to the current directory. When used in file paths, it represents the directory from which the script or application is currently being executed.

For example, if your current working directory is "/home/user/" and you reference a file as "./example.txt," it means "example.txt" in the current directory ("/home/user/").

.. (Double Dot): The double dot (..) refers to the parent directory. When used in file paths, it represents the directory one level up in the hierarchy.

For example, if your current working directory is "/home/user/documents/" and you reference a file as "../example.txt," it means "example.txt" in the parent directory ("/home/user/").

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

Ans - In the path "C:\bacon\eggs\spam.txt," in Python:

Directory Name (dir name): C:\bacon\eggs

Base Name (base name): spam.txt

In this path:

The directory name is the portion that specifies the folder or directory where the file is located. It includes everything up to the last backslash ("").

The base name is the name of the file itself, including its extension. It is everything after the last backslash ("").

So, in the given example:

Directory Name: C:\bacon\eggs

Base Name: spam.txt

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

Three common mode arguments:

Read Mode ('r'):

Used for reading the contents of a file.

If the file does not exist, it raises a FileNotFoundError.

This is the default mode if no mode is specified.

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

content = file.read()

print(content)

Write Mode ('w'):

Used for writing to a file.

If the file exists, it truncates the file to zero length. If the file does not exist, it creates a new file.

It will overwrite the existing content of the file.

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

file.write('Hello, World!')

Append Mode ('a'):

Used for appending data to the end of a file.

If the file does not exist, it creates a new file.

Unlike write mode ('w'), it does not truncate the existing content.

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

file.write('Appending new content.')

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

Ans - If an existing file is opened in write mode ('w') in Python, the following behavior occurs:

If the file exists:

The contents of the existing file will be truncated to zero length, effectively erasing the previous content.

The file is positioned at the beginning, ready for writing.

If the file does not exist:

A new file is created.

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

Ans - In Python, read() and readlines() are both methods used to read content from a file, but they have different behaviors:

read() method:

Reads the entire content of the file as a single string.

Returns a string containing the entire content of the file, including newline characters (\n).

If you do not specify the number of bytes to read, it will read the entire file.

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

content = file.read()

print(content)

readlines() method:

Reads all lines from the file and returns a list where each element represents a line.

The newline characters (\n) are included in each line.

If you do not specify the number of bytes to read, it will read all lines from the current position to the end of the file.

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

lines = file.readlines()

print(lines)

In summary:

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

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, the shelve module provides a persistent, dictionary-like storage system. A shelf value, created using the shelve.open() function, resembles a dictionary. It allows you to store and retrieve key-value pairs persistently.

A shelf is similar to a dictionary in the sense that it is mutable, and you can use keys to access corresponding values. However, unlike dictionaries, a shelf persists its data to disk, making it suitable for storing and retrieving data across different program executions.

import shelve

# Open a shelf file

with shelve.open('myshelf') as shelf:

# Add key-value pairs

shelf['key1'] = 'value1'

shelf['key2'] = [1, 2, 3]

# Retrieve values using keys

value1 = shelf['key1']

value2 = shelf['key2']

print(value1)

print(value2)

In this example, the shelf behaves like a dictionary where 'key1' maps to 'value1' and 'key2' maps to a list [1, 2, 3]. The data is persisted to the 'myshelf' file on disk. The shelf file can be reopened in another execution of the program, and the stored data can be accessed.