1. To what does a relative path refer?

A relative path in computing refers to a file or directory location that is specified relative to the current working directory or the location of the script or program being executed. It is a path that is expressed in relation to another location rather than starting from the root directory.

For example, if the current working directory is /home/user/, a relative path of documents/file.txt would refer to the file file.txt located within the documents directory relative to the current working directory.

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

In most operating systems, an absolute path starts with the root directory. The root directory is the highest-level directory in the file system hierarchy and serves as the starting point for navigating the file system.

On Unix-like systems (such as Linux, macOS, and Unix), the root directory is denoted by a forward slash (/). For example, an absolute path on a Unix-like system may look like /home/user/file.txt.

On Windows systems, the root directory is typically represented by a drive letter followed by a colon (C:, D:, etc.). For example, an absolute path on a Windows system may look like C:\Users\User\file.txt.

Regardless of the operating system, an absolute path provides the complete and unambiguous location of a file or directory within the file system.

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

The functions os.getcwd() and os.chdir() are part of the os module in Python, which provides various functions for interacting with the operating system.

* os.getcwd(): This function returns the current working directory. The current working directory is the directory in which the Python script is currently executing. It represents the path from which relative paths are resolved. The getcwd() function returns a string representing the absolute path of the current working directory.
* os.chdir(path): This function changes the current working directory to the specified path. It takes a string argument path which represents the directory path to which the working directory should be changed. After calling os.chdir(), the current working directory will be updated to the specified path.

Eg:

import os

# Get the current working directory

current\_dir = os.getcwd()

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

# Change the current working directory

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

os.chdir(new\_dir)

print("Changed to new working directory:", os.getcwd())

# Verify the change by listing the files in the new directory

files = os.listdir()

print("Files in the new directory:")

for file in files:

print(file)

4. What are the . and .. folders?

In file systems, the . (dot) and .. (dot dot) folders have special meanings:

1. . (dot): It refers to the current directory. When you use . as a path component, it represents the directory you are currently in.
2. .. (dot dot): It refers to the parent directory. When you use .. as a path component, it represents the directory that contains the current directory.

For example, if you are in the directory /home/user, then . refers to /home/user (the current directory) and .. refers to /home (the parent directory).

5. 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 refers to C:\bacon\eggs, and the base name refers to spam.txt

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

The open() function in Python can be called with different mode arguments to specify how the file should be opened. The three main mode arguments are:

* "r": This mode stands for "read" and is used for opening a file for reading. It is the default mode if no mode is specified. With this mode, you can read the contents of the file but cannot modify it.
* "w": This mode stands for "write" and is used for opening a file for writing. If the file already exists, it will be truncated (emptied) before writing. If the file does not exist, a new file will be created. With this mode, you can write and modify the contents of the file.
* "a": This mode stands for "append" and is used for opening a file for appending data. If the file already exists, the new data will be added at the end of the file. If the file does not exist, a new file will be created. With this mode, you can only append data to the file without modifying the existing content.

Eg:

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

content = file.read()

print(content)

file.close()

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

file.write("This is some content.")

file.close()

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

file.write("This is additional content.")

file.close()

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

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

1. If the file exists, its contents will be truncated or overwritten. All existing content will be removed, and the file will be considered empty.
2. If the file does not exist, a new file with the specified name will be created.

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

read() returns a single string containing the entire file contents, while readlines() returns a list of strings, with each string representing a line from the file.

9. What data structure does a shelf value resemble?

In Python, a shelf value resembles a dictionary data structure. It provides a persistent, dictionary-like object that can be used to store and retrieve key-value pairs. A shelf value allows you to store and retrieve Python objects by using keys, similar to how a dictionary works. The shelf data is stored in a file on the disk, which makes it useful for preserving data across multiple program executions.

You can perform dictionary-like operations on a shelf value, such as accessing values using keys, adding new key-value pairs, modifying existing values, and deleting key-value pairs. However, unlike a dictionary, the values stored in a shelf can be of any picklable data type, as they are serialized and stored in the file.

To use a shelf value, you need to import the shelve module in Python. It provides the necessary functions to open, create, and manipulate shelf values.

Eg:

import shelve

# Open or create a shelf file

shelf\_file = shelve.open('mydata')

# Store data in the shelf

shelf\_file['name'] = 'John Doe'

shelf\_file['age'] = 30

shelf\_file['city'] = 'New York'

# Retrieve and display data from the shelf

print(shelf\_file['name'])

print(shelf\_file['age'])

print(shelf\_file['city'])

# Modify data in the shelf

shelf\_file['age'] = 31

# Delete a key-value pair from the shelf

del shelf\_file['city']

# Close the shelf file

shelf\_file.close()