1. To what does a relative path refer?

Ans: A relative path is a type of path that specifies the location of a file or directory in relation to the current working directory. In other words, a relative path is a path that does not include the full or absolute path to the file or directory, but instead specifies the location of the file or directory relative to the current working directory.

For example, suppose you are in the /home/user directory and you want to access the documents directory inside the /home/user directory. You could specify the full or absolute path to the documents directory as /home/user/documents. However, if you are already in the /home/user directory, you can simply specify the relative path to the documents directory as documents, which will allow you to access the documents directory without needing to specify the full or absolute path.

Relative paths are typically used when you want to access files or directories that are located in the same or a nearby directory as the current working directory. This can make it easier to access these files or directories without needing to specify the full or absolute path, which can be long and difficult to remember.

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

Ans: The specific character that an absolute path starts with can vary depending on the operating system you are using. In general, an absolute path on a Unix-based operating system, such as Linux or macOS, will start with a forward slash (/), which represents the root directory of the file system. On a Windows operating system, an absolute path will typically start with a drive letter followed by a colon (e.g. C:) to specify the drive or partition on which the file or directory is located, followed by a backslash (\) to separate the drive letter from the rest of the path.

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

Ans: The os module in Python provides functions for interacting with the operating system. The os.getcwd() function is used to get the current working directory, which is the directory that the Python interpreter is currently working in. This function returns the absolute path of the current working directory as a string.

The os.chdir() function is used to change the current working directory. This function takes a string argument specifying the path of the directory to which you want to change the current working directory. The function will then change the current working directory to the specified directory, and it will return None.

```
import os

# Print the current working directory.

print(os.getcwd())
```

```
# Change the current working directory to the 'documents' directory

# inside the current working directory.

os.chdir('documents')

# Print the current working directory again, to confirm that it

# has been changed.

print(os.getcwd())
```

4. What are the . and .. folders?

Ans: In many operating systems, including Linux, macOS, and Windows, the . (dot) and .. (dot-dot) directories are special directories that are used to represent the current directory and the parent directory, respectively. These directories are often used in path names to refer to the current or parent directory without needing to specify the full or absolute path to the directory.

The . (dot) directory is used to represent the current directory. This means that if you specify . as part of a path, it will refer to the directory in which the current file or command is located. For example, if you are in the /home/user/documents directory and you specify the path ./file.txt, it will refer to the file.txt file in the /home/user/documents directory.

The .. (dot-dot) directory is used to represent the parent directory. This means that if you specify .. as part of a path, it will refer to the directory that is one level above the current directory in the file system hierarchy. For example, if you are in the /home/user/documents directory and you specify the path ../file.txt, it will refer to the file.txt file in the /home/user directory.

5. 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, the bacon and eggs directories are part of the directory name, while the spam.txt file is the base name.

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

Ans: The open() function in Python is used to open a file and return a file object that can be used to read from or write to the file. The open() function takes a string argument specifying the path of the file to be opened, and it also takes an optional mode argument that specifies how the file should be opened.

There are several different values that can be passed to the mode argument to specify how the file should be opened. The three most common values for the mode argument are:

'r': This mode is used to open a file for reading only. If the file does not exist, an error will be raised.

'w': This mode is used to open a file for writing only. If the file does not exist, it will be created. If the file already exists, it will be overwritten.

'a': This mode is used to open a file for writing only, in append mode. If the file does not exist, it will be created. If the file already exists, new data will be written to the end of the file, without overwriting the existing data.

We can use the 'x' mode to open a file for exclusive creation, or the 'b' mode to open a file in binary mode. We can also combine these modes using the + character to specify different combinations of reading and writing, as well as binary and text modes.

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

Ans: If an existing file is opened in write mode using the 'w' value for the mode argument to the open() function, the file will be overwritten. This means that any existing data in the file will be deleted, and the file will be truncated to zero bytes.

If we open this file in write mode using the following code:

```
f = open('myfile.txt', 'w')
```

then the contents of the file will be deleted, and the file will be truncated to zero bytes. This means that the file will be empty, and any attempts to read from the file will return an empty string.

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

Ans: The read() and readlines() methods are both used to read data from a file object in Python, but they operate in slightly different ways. The read() method reads the entire contents of the file as a single string, while the readlines() method reads the entire contents of the file as a list of strings, where each string corresponds to a single line of the file.

```
# Open a file for reading.

f = open('myfile.txt', 'r')

# Read the entire contents of the file using the read() method.

contents = f.read()

# Print the contents of the file.

print(contents)

# Read the entire contents of the file using the readlines() method.

lines = f.readlines()
```

```
# Print the contents of the file as a list of strings.

print(lines)
```

9. What data structure does a shelf value resemble?

Ans: A shelf value in Python resembles a dictionary data structure. A shelf is a persistent, dictionary-like object that is stored on disk using the shelve module in Python. It provides a simple interface for storing and retrieving data, similar to a dictionary, but the data is stored on disk rather than in memory.

A shelf value can be created by importing the shelve module and calling the open() function, which takes a string argument specifying the name of the shelf file to be created or opened. This function returns a shelf object that can be used to store and retrieve data in the same way as a dictionary.

```
import shelve

# Open a shelf file named 'mydata.db'.

s = shelve.open('mydata.db')

# Store some data in the shelf.

s['key1'] = 'value1'

s['key2'] = 'value2'

# Retrieve some data from the shelf.

print(s['key1'])

print(s['key2'])

# Close the shelf file.

s.close()
```