**Python Assignment 9**

Q1. 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 of the program. It specifies the location of a file or directory relative to the directory in which the Python script is executed.

For example, if the current working directory is /home/user/documents and the file we want to access is located in /home/user/documents/project/files/file.txt, then the relative path to the file from the current working directory would be project/files/file.txt.

Relative paths can be specified using . to refer to the current directory or .. to refer to the parent directory. For example, if we are currently in /home/user/documents/project and want to access a file in the files directory, we can use the relative path ../files/file.txt.

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

Ans. An absolute path in a file system specifies the complete path of a file or directory from the root directory of the file system. The root directory is the starting point of the file system hierarchy and varies based on the operating system.

In Windows, an absolute path typically starts with a drive letter, followed by a colon (:), and then a backslash (\). For example, C:\Users\JohnDoe\Documents\file.txt is an absolute path that refers to a file named file.txt in the Documents folder of the JohnDoe user's home directory on the C: drive.

In Unix-based operating systems (e.g., Linux, macOS), an absolute path starts with a forward slash (/). For example, /home/johndoe/Documents/file.txt is an absolute path that refers to a file named file.txt in the Documents folder of the johndoe user's home directory.

Note that the syntax for absolute paths can vary slightly depending on the operating system and file system.

Q3. What do the functions os.getcwd() and os.chdir() do?

Ans. In Python, os.getcwd() and os.chdir() are functions provided by the os module for interacting with the operating system.

os.getcwd() stands for "get current working directory". This function returns a string that represents the current working directory of the Python script or shell that is being executed. The working directory is the directory that the script is currently running in.

For example:

import os

current\_directory = os.getcwd()

print(current\_directory)

This will print the current working directory.

os.chdir() stands for "change directory". This function is used to change the current working directory of the Python script or shell that is being executed. The new working directory is passed as an argument to the function.

For example:

import os

os.chdir("/path/to/new/directory")

This will change the current working directory to the directory specified in the argument.

Q4. What are the . and .. folders?

Ans. In most file systems, including those used by modern operating systems like Windows, macOS, and Linux, there are two special directories called . and .. that serve important functions.

The . directory, also known as the "current directory", represents the directory you are currently in. It is used to reference files and directories in the current directory. For example, if you are in the directory /home/user/documents, the file example.txt in the same directory can be referred to as ./example.txt.

The .. directory, also known as the "parent directory", represents the directory that contains the current directory. It is used to reference files and directories in the parent directory. For example, if you are in the directory /home/user/documents, the file example.txt in the parent directory can be referred to as ../example.txt.

These directories are important because they allow you to navigate and access files and directories within the file system. They are automatically created in every directory and are always present, even if they are not explicitly shown in directory listings or file managers.

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

Ans. In the file path C:\bacon\eggs\spam.txt:

The directory name is C:\bacon\eggs.

The base name is spam.txt.

The directory name is the path to the directory that contains the file, while the base name is the name of the file itself. The directory name includes all the directories that lead up to the file, separated by the directory separator (\ in this case) in Windows. The base name includes the file name and the file extension (if it has one).

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

Ans. In Python, the open() function is used to open a file and returns a file object. The function can take several arguments, including a filename and a mode. The mode argument specifies the purpose for which the file is opened and determines what operations can be performed on it. There are three main mode arguments that can be passed to the open() function:

'r' - Read mode: This mode is used to open a file for reading. The file pointer is placed at the beginning of the file, and you can read the contents of the file.

'w' - Write mode: This mode is used to open a file for writing. If the file already exists, its contents will be truncated. If the file does not exist, it will be created. You can write data to the file using the file object's write() method.

'a' - Append mode: This mode is used to open a file for appending. The file pointer is placed at the end of the file, and any data you write to the file will be added to the end.

For example, to open a file example.txt in read mode, you can use the following code:

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

Similarly, to open a file in write mode, you can use the following code:

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

And to open a file in append mode, you can use the following code:

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

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

Ans. If an existing file is opened in write mode ('w'), its contents will be truncated or overwritten. In other words, any data that was previously in the file will be deleted, and new data can be written to the file from scratch. If the file doesn't exist, a new file with the specified name will be created.

For example, suppose you have a file example.txt with the following contents:

Hello, world!

If you open this file in write mode and write some new data to it, like this:

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

file.write('Goodbye, world!')

file.close()

The original contents of the file will be completely replaced with the new data. If you open the file again, you'll see that it now contains only the text "Goodbye, world!". It's important to be careful when using write mode, especially with existing files, as you may unintentionally delete important data.

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

Ans. In Python, both the read() and readlines() methods are used to read data from a file, but they return different data types.

The read() method reads the entire contents of a file and returns a string containing all the data. If you specify an integer argument, read() will read that many bytes from the file, starting from the current file position. For example:

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

data = file.read() # read the entire file

print(data)

file.close()

The readlines() method, on the other hand, reads all the lines in the file and returns a list of strings, where each string is a line from the file. For example:

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

lines = file.readlines() # read all lines from the file

print(lines)

file.close()

So, the main difference between read() and readlines() is that read() returns a string with the entire contents of the file, while readlines() returns a list of strings where each string is a line from the file.

In summary, you would use read() when you want to read the entire file as a single string, and readlines() when you want to read the file as a list of strings, where each string is a line from the file.

Q9. What data structure does a shelf value resemble?

Ans. In Python, the shelve module is used for object persistence, which means it allows you to store Python objects in a file on disk, and retrieve them later. The shelve module provides a dictionary-like interface for storing and retrieving objects, with keys that are strings and values that are Python objects.

A shelf value in Python is a dictionary-like object that is used to store and retrieve Python objects. The main difference between a shelf and a regular dictionary is that a shelf is persistent, meaning it can be stored on disk and retrieved later, while a regular dictionary is stored in memory and only exists for the lifetime of the program.

A shelf value in Python resembles a dictionary data structure, where keys are strings and values are Python objects. You can access values in a shelf by key, just like you would in a dictionary:

import shelve

# open the shelf file

my\_shelf = shelve.open('my\_shelf.db')

# add some objects to the shelf

my\_shelf['key1'] = {'name': 'Alice', 'age': 30}

my\_shelf['key2'] = [1, 2, 3, 4, 5]

# retrieve objects from the shelf

obj1 = my\_shelf['key1']

obj2 = my\_shelf['key2']

# close the shelf file

my\_shelf.close()

In this example, my\_shelf is a shelf object that is used to store and retrieve two Python objects, a dictionary and a list. The keys key1 and key2 are strings that are used to identify the objects in the shelf, and the values are the actual Python objects. You can see that accessing values in the shelf is similar to accessing values in a dictionary, using the keys.