**FSDS MAY BATCH 2022(Python Basics 9)**

**Submitted by: Shubham Tiwari**

Q1. To what does a relative path refer?

Ans: A relative path refers to a location of a file or directory relative to the current working directory. In a relative path, the location of the file or directory is specified relative to the current directory, instead of specifying the absolute path from the root directory.

For example, if the current working directory is **C:\bacon\eggs**, a relative path of **spam.txt** refers to the file **C:\bacon\eggs\spam.txt**, and a relative path of **..\spam** refers to the directory **C:\bacon\spam**.

Relative paths are often used in scripts or when working with files and directories, as they are more flexible than absolute paths, as they can be used regardless of the current working directory.

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

Ans: An absolute path starts with the root directory of the file system in most operating systems.

For example, on Unix-based systems like Linux and macOS, an absolute path starts with ‘**/’**, while on Windows-based systems, an absolute path starts with a drive letter, such as ‘**C:\’**.

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

Ans: ‘**os.getcwd()’** returns the current working directory (cwd) of the process as a string.

**‘os.chdir(path)’** changes the current working directory to the given **path**.

Q4. What are the . and .. folders?

Ans: ‘**.’** (single dot) and ‘**..’** (double dots) are special directories in Unix-like operating systems, including Linux and macOS, that refer to the current directory and the parent directory respectively.

**‘./’** refers to the current directory, and ‘**../’** refers to the parent directory of the current directory. These can be used as relative paths when navigating the file system.

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**, **C:\bacon\eggs** is the directory name and **spam.txt** is the base name.

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

Ans: The three "mode" arguments that can be passed to the **open()** function in Python are:

1. **'r'** - read mode (default), opens the file for reading only.
2. **'w'** - write mode, opens the file for writing and truncates the file to zero length or creates a new file if it does not exist.
3. **'a'** - append mode, opens the file for writing and appends to the end of the file if it exists. If the file does not exist, it creates a new file for writing.

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

Ans: If an existing file is opened in write mode (**'w'**) using the **open()** function in Python, the contents of the file will be truncated, meaning all the previous data in the file will be deleted. The file will be opened for writing, and we can write new data to it. If the file does not exist, a new file with the specified name will be created.

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

Ans: **read()** and **readlines()** are two methods in Python that can be used to read data from a file. The main difference between them is the format in which they return the file contents:

* **read()** returns the entire contents of the file as a single string.
* **readlines()** returns the contents of the file as a list of strings, where each string represents a line in the file.

So, if we want to read the entire contents of a file and process it as a single string, you can use **read()**. If we want to read the contents line by line, we can use **readlines()**.

Q9. What data structure does a shelf value resemble?

Ans: A shelf value in Python resembles a dictionary. A shelf is a persistent, dictionary-like object that allows us to store and retrieve data using keys, just like a dictionary. However, unlike a dictionary, a shelf can be stored to and retrieved from disk, so its contents persist even after the Python process that created it has terminated.