**Q1. In Python 3.X, what are the names and functions of string object types?**

Following are the names and function of string object

* Capitalize (): Returns a copy of the string with the first character capitalized and the rest lowercased.
* Lower (): Returns a copy of the string converted to lowercase.
* Upper (): Returns a copy of the string converted to uppercase.
* Swapcase (): Returns a copy of the string with uppercase characters converted to lowercase and vice versa.
* Title (): Returns a copy of the string with the first character of each word capitalized and the rest lowercased.
* Strip (): Returns a copy of the string with leading and trailing whitespace removed.
* Split (): Returns a list of substrings separated by a specified delimiter.
* Join (iterable): Concatenates the elements of an iterable (e.g., a list) into a single string, with each element separated by the string on which the method is called.
* Replace (old, new): Returns a copy of the string with all occurrences of the "old" substring replaced by the "new" substring.
* Startswith(prefix): Checks if the string starts with a specified prefix and returns True or False accordingly.
* endswith(suffix): Checks if the string ends with a specified suffix and returns True or False accordingly.
* isdigit(): Checks if all characters in the string are digits and returns True or False.
* isalpha(): Checks if all characters in the string are alphabetic and returns True or False.
* isalnum(): Checks if all characters in the string are alphanumeric (either letters or digits) and returns True or False.
* len(): Returns the length of the string (number of characters).

**Q2. How do the string forms in Python 3.X vary in terms of operations?**

**Q3. In 3.X, how do you put non-ASCII Unicode characters in a string?**

If the file is encoded with Unicode you can use Unicode escape character followed by ‘u’ and a hexadecimal Unicode number for non-ASCII Unicode character or using ‘u’ as prefix before non-ASCII Unicode.

**Q4. In Python 3.X, what are the key differences between text-mode and binary-mode files?**

In text mode the file is data is handled as string data and hence needs to be encoded and decode where as in binary-mode the data is handle in raw bytes that is 0 and 1 so need of encoding and decoding.

**Q5. How can you interpret a Unicode text file containing text encoded in a different encoding than your platform's default?**

At first it is necessary to check the encoding of the data for that the official document or the source of the data is to be checked. Then open the file using with open keywords and give encoding as argument and by using read command we can interpret the text file.

**Q6. What is the best way to make a Unicode text file in a particular encoding format?**

We have to use ‘with Open() ‘ keywords in **write** format and give encoding as an argument in which you want to make text file.

**Q7. What qualifies ASCII text as a form of Unicode text?**

ASCII text is considered a form of Unicode text because the first 128 characters of the Unicode character set (U+0000 to U+007F) directly correspond to the ASCII characters.

**Q8. How much of an effect does the change in string types in Python 3.X have on your code?**

Here are some key effects of the string type change in Python 3.x:

* Encoding and Decoding: In Python 3.x, string objects (str) are Unicode strings by default. This means that handling text data is more straightforward as you don't need to explicitly decode or encode strings for Unicode support. However, if you were relying on byte strings (str in Python 2.x) with specific encodings, you may need to update your code to handle encoding and decoding appropriately.
* Print Statements: The print statement in Python 2.x implicitly converted non-ASCII characters to the default encoding before outputting. In Python 3.x, you need to handle encoding explicitly when printing non-ASCII characters. You can use the encode() method to encode the string before printing.
* Method and Function Changes: Python 3.x introduced new methods and functions related to string manipulation. For example, the unicode() function in Python 2.x, used to convert byte strings to Unicode strings, is not available in Python 3.x. You may need to update such usages to accommodate the new string methods and functions available in Python 3.x.
* Byte Strings: In Python 2.x, byte strings (str) were used to represent binary data. In Python 3.x, the bytes type is used explicitly for binary data, while string literals (str) are Unicode strings. If your code relied on byte strings for binary data operations, you'll need to adjust your code to use the bytes type appropriately.
* Comparison Operators: The behavior of comparison operators (<, >, <=, >=) for strings changed in Python 3.x due to the use of Unicode strings. The ordering of strings with different encodings can produce different results. It's important to review and update any code that relies on string comparisons.