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

Sol. Python 3.0 has three string types: str (for Unicode text, including ASCII), bytes (for binary data with absolute byte values), and bytearray (a mutable flavor of bytes). The str type usually represents content stored on a text file, and the other two types generally represent content stored on binary files.

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

Sol. Python 2.6 has two main string types: str (for 8-bit text and binary data) and unicode (for wide-character text). The str type is used for both text and binary file content; unicode is used for text file content that is generally more complex than 8 bits. Python 2.6 (but not earlier) also has 3.0’s bytearray type, but it’s mostly a back-port and doesn’t exhibit the sharp text/binary distinction that it does in 3.0.

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

Sol. Non-ASCII Unicode characters can be coded in a string with both hex (\xNN) andUnicode (\uNNNN, \UNNNNNNNN) escapes. On some keyboards, some non-ASCII characters— certain Latin-1 characters, for example—can also be typed directly.

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

Sol. In 3.0, text-mode files assume their file content is Unicode text (even if it’s ASCII) and automatically decode when reading and encode when writing. With binarymode files, bytes are transferred to and from the file unchanged. The contents of text-mode files are usually represented as str objects in your script, and the contents of binary files are represented as bytes (or bytearray) objects. Text-mode files also handle the BOM for certain encoding types and automatically translate endof- line sequences to and from the single \n character on input and output unless this is explicitly disabled; binary-mode files do not perform either of these steps.

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

Sol. To read files encoded in a different encoding than the default for your platform, simply pass the name of the file’s encoding to the open built-in in 3.0 (codecs.open() in 2.6); data will be decoded per the specified encoding when it is read from the file. You can also read in binary mode and manually decode the bytes to a string by giving an encoding name, but this involves extra work and is somewhat error-prone for multibyte characters (you may accidentally read a partial character sequence).

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

Sol. To create a Unicode text file in a specific encoding format, pass the desired encodingname to open in 3.0 (codecs.open() in 2.6); strings will be encoded per the desired encoding when they are written to the file. You can also manually encodea string to bytes and write it in binary mode, but this is usually extra work.

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

Sol. ASCII text is considered to be a kind of Unicode text, because its 7-bit range of values is a subset of most Unicode encodings. For example, valid ASCII text is alsovalid Latin-1 text (Latin-1 simply assigns the remaining possible values in an 8-bit byte to additional characters) and valid UTF-8 text (UTF-8 defines a variable-byte scheme for representing more characters, but ASCII characters are still represented with the same codes, in a single byte).

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

Sol. The impact of Python 3.0’s string types change depends upon the types of strings you use. For scripts that use simple ASCII text, there is probably no impact at all: the str string type works the same in 2.6 and 3.0 in this case. Moreover, although string-related tools in the standard library such as re, struct, pickle, and xml maytechnically use different types in 3.0 than in 2.6, the changes are largely irrelevantto most programs because 3.0’s str and bytes and 2.6’s str support almost identical interfaces. If you process Unicode data, the toolset you need has simply moved from 2.6’s unicode and codecs.open() to 3.0’s str and open. If you deal with binary data files, you’ll need to deal with content as bytes objects; since they have a similar interface to 2.6 strings, though, the impact should again be minimal