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

Answer:

Python 3.X has following string object types:

1. str 🡪 this is for representing Unicode text; str() function uses to create string objects and this function converts the specified value into a string.
2. bytes 🡪 this is for representing binary data which also include encoded text; bytes() function uses to create a bytes object of any given size and it also uses to convert a string to bytes while passing the string as first input and encoding as the second input to the function.
3. bytearray 🡪 this is a variant of bytes and it is mutable; bytearray objects can be created by calling the bytearray() function and it returns an array of bytes of the given size.

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

Answer:

Strings in Python are arrays of bytes representing Unicode characters. Python supports ASCII as a subset of Unicode and the default encoding of characters in Python is UTF-8. Python 3.X supports following string object types to form string with similar operation types while they have different roles: str, bytes, and bytearray.

🡪 str: this is the Unicode string type and it supports string operations like slicing, concatenation, formatting, searching, etc.

🡪 bytes: this is the raw bytes type that supports only a few set of operations, such as slicing, concatenation, and searching. It is more memory-efficient than the str type.

🡪 bytearray: this is a mutable and a variant of bytes type that supports all the same operations as bytes but also allows the modification of individual elements.

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

Answer:

In Python 3.X, we can include non-ASCII Unicode characters in a string by adding a “u” character prefix before the string literal.

Example:

str1 = u"\u00f8"

print(str1)

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

Answer:

|  |  |
| --- | --- |
| Text-mode files | Binary-mode files |
| This mode is the default mode, and it is used for reading and writing text files | This mode is used for reading and writing binary files |
| Handle data as strings of Unicode characters | Handle data as a sequence of bytes |
| Use the platform-specific newline convention when writing, and translate it to the universal newline convention when reading | Do not perform any newline translation |
| Automatically encode Unicode strings to bytes when writing, and decode bytes to Unicode strings when reading , using the specified encoding | Do not perform any encoding or decoding |
| Provide additional high-level I/O methods for working with strings, such as readline() and writelines() | Provide lower-level I/O methods that work with bytes, such as read() and write() |
| It may raise exceptions when encountering encoding or decoding errors | It may raise exceptions when encountering I/O errors or reaching the end of file |
| Text files are files that contain text data, such as strings or characters. | Binary files, on the other hand, are files that contain non-text data, such as images, audio files, and executable files. |

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

Answer:

A Unicode text file that containing encoded text in a different encoding can be interpreted by using the specific encoding parameter with the open() function.

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

Answer:

Write the Unicode text file with open() function in Python 3.x while filename and particular encoding format (say, UTF-8) needs to be used as an arguments or parameters of the function.

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

Answer:

Generally, ASCII encodes digits, letters, symbols, etc., whereas Unicode encodes special texts from letters, symbols, different languages, etc. ASCII is basically a subset of the Unicode encoding scheme and Unicode is a superset of ASCII. Perhaps, all ASCII characters are included in the Unicode. Hence, any text that contains ASCII characters are valid Unicode text.

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

Answer:

In Python 3.x, all strings are Unicode by default and only thing is that we need to use different string methods and encoding or decoding functions to work with it in a Unicode context. Hence there will be no such effect on the code.