**ADITYA AMIN**

**ASSIGN : 09**

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

str (Unicode String):

The str type represents Unicode strings in Python 3.x. It is the primary string object type used for handling text.

String literals in Python are represented using the str type by default. For example: "Hello, World!".

The str type provides various methods for manipulating and working with strings, such as lower(), upper(), split(), join(), startswith(), endswith(), and many more.

Example usage: my\_string = "Hello, World!".

bytes (Byte String):

The bytes type represents sequences of bytes in Python 3.x. It is used for handling binary data, such as raw data or encoded text.

Byte strings are denoted by a prefix b before the string literal. For example: b"Hello, World!".

The bytes type provides methods for encoding and decoding strings using various character encodings, such as encode(), decode(), hex(), fromhex(), etc.

Example usage: my\_bytes = b"Hello, World!".

bytearray (Mutable Byte String):

The bytearray type is similar to bytes but provides a mutable sequence of bytes. It allows modifications to the individual elements of the byte string.

Byte arrays are denoted by a prefix bytearray before the string literal. For example: bytearray(b"Hello, World!").

The bytearray type provides methods for modifying and manipulating byte arrays, such as append(), insert(), pop(), remove(), extend(), etc.

Example usage: my\_bytearray = bytearray(b"Hello, World!").

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

str (Unicode String):

The str type is used for handling Unicode strings and supports a wide range of string operations.

It supports string concatenation using the + operator, string repetition using the \* operator, and string indexing and slicing.

bytes (Byte String):

The bytes type is used for handling sequences of bytes, typically representing binary data or encoded text.

Byte strings support similar operations as str, including concatenation, repetition, and indexing. However, byte strings are immutable as well.

bytearray (Mutable Byte String):

The bytearray type is similar to bytes but provides a mutable sequence of bytes, allowing modifications.

Byte arrays support all the operations of bytes, including concatenation, repetition, indexing, and slicing. Additionally, they allow in-place modifications like assignment to a specific index.

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

Unicode Escape Sequences:

Unicode escape sequences allow you to represent non-ASCII Unicode characters using their hexadecimal code points.

The escape sequence consists of the \u prefix followed by four hexadecimal digits representing the Unicode code point of the character.

Direct Entry in Source Code (UTF-8 Encoding):

If your source code file is encoded in a compatible encoding like UTF-8, you can directly include non-ASCII characters in the string.

Simply enter the characters directly into the string, and Python will interpret them as Unicode characters.

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

Text-mode files:

Text-mode files are opened using the 't' or 'text' mode when using the open() function.

In text-mode, the file data is treated as a sequence of Unicode characters.

When reading from a text-mode file, Python automatically performs decoding, converting the bytes read from the file into Unicode characters based on the specified encoding (or the default system encoding if not explicitly specified).

When writing to a text-mode file, Python automatically performs encoding, converting the Unicode characters into bytes before writing them to the file using the specified encoding (or the default system encoding if not explicitly specified).

Text-mode files are suitable for working with textual data, such as plain text files, CSV files, JSON files, etc.

Example: file = open('text\_file.txt', 'rt')

Binary-mode files:

Binary-mode files are opened using the 'b' or 'binary' mode when using the open() function.

In binary-mode, the file data is treated as a sequence of bytes.

When reading from a binary-mode file, data is read as-is without any decoding. The read operation returns the raw bytes from the file.

When writing to a binary-mode file, you need to provide data as bytes, and no automatic encoding or decoding is performed.

Binary-mode files are suitable for working with non-textual data, such as images, audio files, video files, or any file format that requires exact byte representation.

Example: file = open('binary\_file.bin', 'rb')

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

To interpret a Unicode text file containing text encoded in a different encoding than your platform's default, you can specify the desired encoding explicitly when opening the file using the open() function in Python. This allows you to read the file using the correct encoding and ensure proper decoding of the text.

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

The best way to create a Unicode text file in a particular encoding format is to use the open() function in Python and specify the desired encoding when opening the file for writing.

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

ASCII text is considered a form of Unicode text because ASCII characters are included in the Unicode character set. Unicode provides a broader encoding standard that encompasses ASCII and other characters from various writing systems.

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