Python - Unicode System

What is Unicode System?

Software applications often require to display messages output in a variety in different languages such as in English, French, Japanese, Hebrew, or Hindi. Python's string type uses the Unicode Standard for representing characters. It makes the program possible to work with all these different possible characters.

A character is the smallest possible component of a text. 'A', 'B', 'C', etc., are all different characters. So are 'È' and 'Í'. A unicode string is a sequence of code points, which are numbers from 0 through 0x10FFFF (1,114,111 decimal). This sequence of code points needs to be represented in memory as a set of code units, and code units are then mapped to 8-bit bytes.

Character Encoding

A sequence of code points is represented in memory as a set of code units, mapped to 8-bit bytes. The rules for translating a Unicode string into a sequence of bytes are called a character encoding.

Three types of encodings are present, UTF-8, UTF-16 and UTF-32. UTF stands for **Unicode Transformation Format**.

Learn **Python** in-depth with real-world projects through our **Python certification course**. Enroll and become a certified expert to boost your career.

Python's Unicode Support

Python 3.0 onwards has built-in support for Unicode. The **str** type contains Unicode characters, hence any string created using single, double or the triple-quoted string syntax is stored as Unicode. The default encoding for Python source code is UTF-8.

Hence, string may contain literal representation of a Unicode character (3/4) or its Unicode value (\u00BE).

Example

</> Open Compiler

```
var = "3/4"
print (var)
```

```
var = "\u00BE"
print (var)
```

This above code will produce the following **output** –

```
3/4
¾
```

Example

In the following example, a string '10' is stored using the Unicode values of 1 and 0 which are \u0031 and u0030 respectively.

</> Open Compiler

```
var = "\u0031\u0030"
print (var)
```

It will produce the following **output** –

```
10
```

Strings display the text in a human-readable format, and bytes store the characters as binary data. Encoding converts data from a character string to a series of bytes. Decoding translates the bytes back to human-readable characters and symbols. It is important not

to confuse these two methods. encode is a string method, while decode is a method of the Python byte object.

Example

In the following example, we have a string variable that consists of ASCII characters. ASCII is a subset of Unicode character set. The encode() method is used to convert it into a bytes object.

</> Open Compiler

```
string = "Hello"
tobytes = string.encode('utf-8')
print (tobytes)
string = tobytes.decode('utf-8')
print (string)
```

The decode() method converts byte object back to the str object. The encodeing method used is utf-8.

```
b'Hello'
Hello
```

Example

In the following example, the Rupee symbol (₹) is stored in the variable using its Unicode value. We convert the string to bytes and back to str.

</> Open Compiler

```
string = "\u20B9"
print (string)
tobytes = string.encode('utf-8')
print (tobytes)
string = tobytes.decode('utf-8')
print (string)
```

When you execute the above code, it will produce the following **output** –

```
₹
b'\xe2\x82\xb9'
₹
```