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

ANS:

|  |  |
| --- | --- |
| Method | True if |
| str.isnumeric() | String consists of only numeric characters |
| str.isspace() | String consists of only whitespace characters |
| str.istitle() | String is in title case |
| str.isupper() | String's alphabetic characters are all upper case |

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

ANS:

Strings are amongst the most popular types in Python. We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes. Creating strings is as simple as assigning a value to a variable. For example −

var1 = 'Hello World!'

var2 = "Python Programming"

## Accessing Values in Strings

Python does not support a character type; these are treated as strings of length one, thus also considered a substring.

To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring. For example −

[Live Demo](http://tpcg.io/53XKOs)

#!/usr/bin/python3

var1 = 'Hello World!'

var2 = "Python Programming"

print ("var1[0]: ", var1[0])

print ("var2[1:5]: ", var2[1:5])

When the above code is executed, it produces the following result −

var1[0]: H

var2[1:5]: ytho

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

ANS:

I have a string that looks like so:

6Â 918Â 417Â 712

The clear cut way to trim this string (as I understand Python) is simply to say the string is in a variable called s, we get:

s.replace('Â ', '')

Here's the code, it really is just the same as above, but now it's in context. The file is saved as UTF-8 in notepad and has the following header:

#!/usr/bin/python2.4

# -\*- coding: utf-8 -\*-

The code:

f = urllib.urlopen(url)

soup = BeautifulSoup(f)

s = soup.find('div', {'id':'main\_count'})

#making a print 's' here goes well. it shows 6Â 918Â 417Â 712

s.replace('Â ','')

save\_main\_count(s)

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

ANS:

There are mainly two types of data files — text file and binary file. A text file consists of human readable characters, which can be opened by any text editor. On the other hand, binary files are made up of non-human readable characters and symbols, which require specific programs to access its contents.

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

ANS:

In text processing, Unicode takes the role of providing a unique code point—a number, not a glyph—for each character. In other words, Unicode represents a character in an abstract way and leaves the visual rendering (size, shape, font, or style) to other software, such as a web browser or word processor.

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

ANS:

 in general, use UTF-8 for writing to files (we don't even have to worry about byte-order with utf-8). utf-8 is the most modern and universally usable encoding - it works in all web browsers, most text-editors (see your settings if you have issues) and most terminals/shells.

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

ANS:

ASCII encodes symbols, digits, letters, etc., whereas Unicode encodes special texts from different languages, letters, symbols, etc. It can be said that ASCII is a subset of the Unicode encoding scheme.

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

ANS:

 Python 3.0 is the mutation of string object types. In a nutshell, 2.X's str and unicode types have morphed into 3.X's bytes and str types, and a new mutable bytearray type has been added. Especially if you process data that is either Unicode or binary in nature, this can have substantial impacts on your code. As a general rule of thumb, how much you need to care about this topic depends in large part upon which of the following categories you fall into:

1. If we deal with non-ASCII Unicode text—for instance, in the context of internationalized applications, Internet content, or XML parsers—you will find support for text encodings to be different in 3.0, but also probably more direct, accessible, and seamless than in 2.6, thanks to 3.0's all-Unicode str.
2. If we deal with binary data—for example, in the form of image or audio files, network transfers, or packed data processed with the struct module—you will need to understand 3.0's new bytes object, and its different and sharper distinction between text and binary data and files.