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

string.Isdecimal	Returns true if all characters in a string are decimal
String.Isalnum	Returns true if all the characters in a given string are alphanumeric.
string.Istitle	Returns True if the string is a titlecased string
String.partition	splits the string at the first occurrence of the separator and returns a tuple.
String.Isidentifier	Check whether a string is a valid identifier or not.
String.len	Returns the length of the string.
String.rindex	Returns the highest index of the substring inside the string if substring is found.
String.Max	Returns the highest alphabetical character in a string.
String.min	Returns the minimum alphabetical character in a string.
String.splitlines	Returns a list of lines in the string.
string.capitalize	Return a word with its first character capitalized.
string.expandtabs	Expand tabs in a string replacing them by one or more spaces
string.find	Return the lowest indexin a sub string.
string.rfind	find the highest index.
string.count	Return the number of (non-overlapping) occurrences of substring sub in string
string.lower	Return a copy of s, but with upper case letters converted to lower case.
string.split	Return a list of the words of the string,If the optional second argument sep is absent or None
string.rsplit()	Return a list of the words of the string s, scanning s from the end.

rpartition()	Method splits the given string into three parts
string.splitfields	Return a list of the words of the string when only used with two arguments.
string.join	Concatenate a list or tuple of words with intervening occurrences of sep.
string.strip()	It return a copy of the string with both leading and trailing characters removed
string.lstrip	Return a copy of the string with leading characters removed.
string.rstrip	Return a copy of the string with trailing characters removed.
string.swapcase	Converts lower case letters to upper case and vice versa.
string.translate	translate the characters using table
string.upper	lower case letters converted to upper case.
string.ljust	left-justify in a field of given width.
string.rjust	Right-justify in a field of given width.
string.center()	Center-justify in a field of given width.
string-zfill	Pad a numeric string on the left with zero digits until the given width is reached.
string.replace	Return a copy of string s with all occurrences of substring old replaced by new.
string.casefold()	Returns the string in lowercase which can be used for caseless comparisons.
string.encode	Encodes the string into any encoding supported by Python.Default encoding is utf-8.
string.maketrans	Returns a translation table usable for str.translate()

Q2. How do the string forms in Python 3.X vary in terms of operations?
Answer : Python 3 default storing of strings is Unicode whereas Python 2 stores need to define Unicode string value with "u."

Q3. In 3.X, how do you put non-ASCII Unicode characters in a string?
Answer: unidecode() method from unidecode library can be used to put non-ASCII Unicode Characters in a string.
E.g.

```
from unidecode import unidecode
print(unidecode(u'ko\u017eu\u0161\u010dek'))
```

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

When a file is opened in ***text mode***, reading its data automatically decodes its content (per a platform default or a provided encoding), and returns it as a `str`; writing takes a `str`, and automatically encodes it before transferring to the file. Text mode files also support universal end-of-line translation, and encoding specification arguments.

When a file is opened in ***binary mode*** by adding a "b" to the mode string argument in the `open()` call, reading its data does not decode it in any way, and simply returns its content raw and unchanged, as a `bytes` object; writing takes a `bytes` object and transfers it to the file unchanged. Binary-mode files also accept a `bytearray` object for the content to be written to the file.

Q5. How can you interpret a Unicode text file containing text encoded in a different encoding than your platform's default?
Answer: Use of encode() and decode() method can be used to you interpret a Unicode text file containing text encoded in a different encoding than your platform's default, by default encoding parameter is “UTF-8”.

Q6. What is the best way to make a Unicode text file in a particular encoding format?
Answer: Use str.encode() and file.write(to make a Unicode text file in a particular encoding format, default encoding format is “UTF-18”.
Call str.encode(encoding) with encoding set to "utf8" to encode str.
Call open(file, mode) to open a file with mode set to "wb" . "wb" writes to files in binary mode and preserves UTF-8 format.
Call file.write(data) to write data to the file.

E.g.

```
unicode_text = u'ƶʒʒʔʕɕʊʁɷʜjʞ'
encoded_unicode = unicode_text.encode("utf8")

a_file = open("textfile.txt", "wb")
a_file.write(encoded_unicode)

a_file = open("textfile.txt", "r")
r reads contents of a file
```

```
contents = a_file.read()

print(contents)
```

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

Answer : Unicode represents most written languages in the world. ASCII has its equivalent in Unicode. The difference between ASCII and Unicode is that ASCII represents lowercase letters (a-z), uppercase letters (A-Z), digits (0–9) and symbols such as punctuation marks while Unicode represents letters of English, Arabic, Greek etc. mathematical symbols, historical scripts, emoji covering a wide range of characters than ASCII.

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

Answer : Python 3 stores strings as Unicode whereas Python 2 requires you to mark a string with a "u" if you want to store it as Unicode. Unicode strings are more versatile than ASCII strings, which are the Python 2 default, as they can store letters from foreign languages as well as emoji and the standard Roman letters and numerals.