**FSDS MAY BATCH 2022(Python Assignment -9)**

**Submitted by: Shubham Tiwari**

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

Ans: The names and functions of string object types are:

capitalize() : Converts the first character of the string to a capital (uppercase) letter

casefold(): Implements caseless string matching

center(): Pad the string with the specified character.

count(): Returns the number of occurrences of a substring in the string.

encode(): Encodes strings with the specified encoded scheme

endswith(): Returns “True” if a string ends with the given suffix

expandtabs(): Specifies the amount of space to be substituted with the “\t” symbol in the string

find(): Returns the lowest index of the substring if it is found

format(): Formats the string for printing it to console

formatmap(): Formats specified values in a string using a dictionary

index(): Returns the position of the first occurrence of a substring in a string

isalnum(): Checks whether all the characters in a given string is alphanumeric or not

isalpha(): Returns “True” if all characters in the string are alphabets

isdecimal(): Returns true if all characters in a string are decimal

isdigit(): Returns “True” if all characters in the string are digits

isidentifier(): Check whether a string is a valid identifier or not

islower(): Checks if all characters in the string are lowercase

isnumeric(): Returns “True” if all characters in the string are numeric characters

isprintable(): Returns “True” if all characters in the string are printable or the string is empty

isspace(): Returns “True” if all characters in the string are whitespace characters

istitle(): Returns “True” if the string is a title cased string

isupper(): Checks if all characters in the string are uppercase

join(): Returns a concatenated String

ljust(): Left aligns the string according to the width specified

lower(): Converts all uppercase characters in a string into lowercase

lstrip(): Returns the string with leading characters removed

maketrans(): Returns a translation table

partition(): Splits the string at the first occurrence of the separator

replace(): Replaces all occurrences of a substring with another substring

rfind(): Returns the highest index of the substring

rindex(): Returns the highest index of the substring inside the string

rjust() :Right aligns the string according to the width specified

rpartition(): Split the given string into three parts

rsplit(): Split the string from the right by the specified separator

rstrip(): Removes trailing characters

splitlines(): Split the lines at line boundaries

startswith(): Returns “True” if a string starts with the given prefix

strip(): Returns the string with both leading and trailing characters

swapcase(): Converts all uppercase characters to lowercase and vice versa

title(): Convert string to title case

translate(): Modify string according to given translation mappings

upper(): Converts all lowercase characters in a string into uppercase

zfill(): Returns a copy of the string with ‘0’ characters padded to the left side of the string

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

Ans: Python Strings are immutable, it means once we declare a string, we can’t modify it. Following are the common string operations that can be performed in Python:

* Concatenation of two or more strings.
* Extracting or slicing partial strings from string values.
* Adding or removing spaces.
* Converting to lower or upper case.
* Formatting strings using string formatters.
* Finding and/or replacing a text in the given string with some other text.

**String Operations can be done in three ways:**

1)Using f-strings

2)By format() method

3)Using % operator

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

Ans: Python allows Unicode string literals to be specified by adding a u character prefix before the string literal:

Scripting example showing Japanese characters. The node that is created has the correct label.

This will create a Unicode string and the label will be appear correctly.

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

Ans: A **text file** is the one in which data is stored in the form of ASCII characters and is normally used for storing a stream of characters. Text files are organized around lines, each of which ends with a newline character (‘\n’). The source code files are themselves text files.

A **binary file** is the one in which data is stored in the file in the same way as it is stored in the main memory for processing. It is stored in binary format instead of ASCII characters. It is normally used for storing numeric information (int, float, double). Normally a binary file can be created only from within a program and its contents can be read only by a program.

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

Ans: The file opened by codecs.open is a file that takes unicode data, encodes it in iso-8859-1 and writes it to the file. However, what we try to write isn’t unicode; you take unicode and encode it in iso-8859-1 ourself. That’s what the unicode.encode method does, and the result of encoding a unicode string is a bytestring (a str type.)

We should either use normal open() and encode the unicode ourself, or (usually a better idea) use codecs.open() and not encode the data ourself.

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

Ans: We should either use normal open() and encode the unicode ourself, or (usually a better idea) use codecs.open() and not encode the data ourself.

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

Ans: ASCII uses 8 bits to encode any character, most of them from the English language used in modern-day programming. It is also used in graphic arts to represent clip art or images using characters.

The major disadvantage of ASCII is that it can represent only 256 different characters as it can use only 8 bits. ASCII cannot be used to encode the many types of characters found around the world. Unicode was extended further to UTF-16 and UTF-32 to encode the various types of characters. Therefore, the significant difference between ASCII and Unicode is the number of bits used to encode.

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

Ans: Change in string types obviously effect our code :

1) Python 3.x introduced some Python 2-incompatible keywords and features that can be imported via the in-built \_\_future\_\_ module in Python 2. It is recommended to use \_\_future\_\_ imports it if we are planning Python 3.x support for our code. For example, if we want Python 3.x’s integer division behavior in Python 2, we can import it via:

from \_\_future\_\_ import division

2) Python 2 doesn’t have a problem with additional parantheses, but in contrast, Python 3.X would raise a SyntaxError if we called the print function the Python 2-way without the parentheses.

3) Python 2 has ASCII str() types, separate unicode(), but no byte type.

Now, in Python 3.X, we finally have Unicode (utf-8) strings, and 2 byte classes: byte and bytearrays.