Learning Quiz 34: Character Encodings and Strings

Due Nov 27 at 1pm **Points** 5 **Questions** 5 **Available** until Dec 4 at 11:59pm **Time Limit** None
Allowed Attempts Unlimited

Instructions

Prior to completing this quiz, be sure to read:

- Section 6.3: Character Encodings and Strings (p. 181-186)

Please also go over Practice Problems 6.7 and 6.8 in the textbook (solutions at the end of the chapter) before attempting this quiz.

This quiz was created for learning purposes. You may attempt this quiz as many times as you would like. The highest score *prior to the deadline* will count towards the final course grade. No late submissions will be accepted.

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	10 minutes	5 out of 5

Score for this attempt: **5** out of 5
Submitted Nov 29 at 11:12am
This attempt took 10 minutes.

Question 1

1 / 1 pts

Strings appear straightforward, but they can actually be fairly messy. This is because computers deal with bits and bytes, and string values need to somehow be encoded with bits and bytes. In earlier sections, working with strings appear intuitive. However, when considering the World Wide Web, characters from other writing systems, and symbols from various fields of study (e.g. math), it's important to understand how strings are represented.

ASCII defines a numeric code for 128 characters and symbols (including backspace, tab, and new line) in the American English language. Table 6.4 (ASCII Encoding) on page 182 in the textbook lists the printable ASCII characters and their corresponding decimal code.

To get the corresponding decimal code of any ASCII character, you can use the ord() method. Here are a few examples:

```
ord('p')
ord('B')
ord('\t')       ##tab
ord('\n')       ##new line
ord('!')
```

Use ord() to determine the ASCII code of the following characters:

Correct! **ord('\a')** 7

Correct! **ord('a')** 97

Correct! **ord('#')** 35

Correct! **ord('5')** 53

Correct! **ord('=')** 61

Other Incorrect Match Options:

- 65
- 127
- 47
- 6
- 5

Question 2

1 / 1 pts

However, the sequence of codes are actually stored in binary. To get the binary encoding, you'll need to convert the decimal value to binary.

Remember, you can use format() to get binary, decimal, hexadecimal, and other forms of a *number* (see Section 4.2, page 104 in textbook). Try printing some of the below examples:

```
'{:b}'.format(1)
'{:b}'.format(2)
'{:b}'.format(3)
'{:b}'.format(4)
'{:b}'.format(5)
'{:b}'.format(6)
'{:b}'.format(7)
'{:b}'.format(8)
'{:b}'.format(100)
'{:b}'.format(126)
```

Note that a byte is eight bits. So the number 3 would be represented as '00000011' (eight digits) in byte (since binary is just '11').

Convert the following decimal values to their binary equivalent:

Correct! **99** 01100011

Correct! **101** — 01100101

Correct! **55** — 00110111

Correct! **111** — 01101111

Other Incorrect Match Options:

- 00111111
- 01111111
- 01000111
- 01100111
- 00110011

Question 3

1 / 1 pts

Method format() does not work for converting string to binary. To get the binary form of a string, you can convert the decimal ASCII value first:

```
'{:b}'.format(ord('a'))
'{:b}'.format(ord('F'))
```

Convert the following string to their equivalent binary values using ord() and format():

Correct! **G** — 01000111

Correct! **k** — 01101011

Correct! **w** — 01110111

Correct! **$** — 00100100

Other Incorrect Match Options:

- 01010111
- 01101010
- 01110011
- 01001000

Question 4

1 / 1 pts

A byte stores eight 0's and 1's. It is common to use hex ASCII codes as shorthand for ASCII binary codes. To get the hex equivalent of a byte, split the byte in half (4 bits and 4 bits). For example, 00100110 will split into 0010 (which is 2) and 0110 (which is 6). The hex equivalent would be 26.

You can easily get the hex value of a number using format(). If you'd like to convert string characters, use the combination of ord() and format():

```
'{:x}'.format(121)
'{:x}'.format(125)        'note this is 01111101 in binary which converts to 7d in hex'
'{:x}'.format(ord('a'))
'{:x}'.format(ord('F'))
```

Convert the following **string** to their equivalent hex value using ord() and format().

Correct!	**'7'**	37
Correct!	**'y'**	79
Correct!	**'T'**	54
Correct!	**'*'**	2a

Other Incorrect Match Options:

- 7
- 20
- 74
- 7d

Question 5

1 / 1 pts

What about characters not in the American English system? ASCII is an American standard and does not cover characters in other languages.

Unicode was developed to be the universal character-encoding scheme, covering characters in all written languages--modern and ancient. In unicode, every character is represented by an code point. The code point is not necessarily the byte representation, but rather a identifier for the particular character.

To access a unicode character, type '\u' in front of the code point. A list of unicode characters can be easily found on the World Wide Web. Here are a few examples:

```
'\u0393'
'\u597d'
'\u007a'
```

While some of us won't need to work with different encodings in our regular programming work, it's important to keep these concepts in mind in case we one day need to access files and pages from

different sources.

Fill in the blanks with one character each to print the Greek letter alpha.

print(' \ u 03b1')

Answer 1:

Correct! \

Answer 2:

Correct! u

Quiz Score: **5** out of 5