Exercise 4 – Solutions

1. Iterate through the character string and apply a bitwise OR with bit mask 0x20 to each character.

   In Python, for example:

```
S = "MyString"
out = ""
for c in S:
    out += chr(ord(c) | 0x20)
print(out)
```

2. 8-bit characters effectively use 7 bit freely, i.e. there are $2^7 = 128$ character codes.
   16-bit " " " 14 " ", i.e. " " $2^{14} = 16384$ " "

   ⇒ The total is $16384 + 128 = 16512$ character codes.

3. • ASCII (quite common legacy encoding) is a valid subset of UTF-8.
   • Substantially more code points than UTF-16. (But currently, there is still a lot of room even in UTF-16.)
   • More efficient on average for text with common markup (e.g. HTML) which is predominantly ASCII. Thus, Western languages are coded much more efficiently and even East Asian languages with lots of surrogate characters benefit in practical contexts such as full web pages.
   • The raw byte stream does not contain 0x00-bytes, so can be handled safely by (legacy) C-style string processing routines.

4. • Iterate through the bytes of the byte string
   • Check the first bit of every byte
     * if set (on at least one byte), it's not ASCII
     * if not set on any byte, it's valid ASCII