Project 6: Base64

Alenn Wright
October 15th, 2021
CS 200

Introduction

For this project we are tasked to write a program or programs that convert a message to base 64 and then back to the message. This means we will have to take a message coded in ASCII and convert that into binary, then divide them into 6 bit sections. From these 6 bit portions we will convert them into decimal and then into the base 64 counterpart. The reverse is in another program. For the base 64 characters we are given a list of the representation and a sample output of the base64 encoding script.

The base64 symbols representing digits values 0 to 63 are as follows:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
```

The sample output is as follows

Original Data:

```
Original data:  Zork

ASCII codes:    Z = $5A  o = $6F  r = $72    k = $6B
In binary:      01011010 01101111 01110010   01101011
```

Regroup as sets of 6 bits and convert to appropriate base64 digit:

```
Groups of 8:    01011010 01101111 01110010   01101011

Groups of 6:    01011010 01101111 01110010   01101011

                 010110 100110 111101 110010
                 011010 11xxxx xxxxxx xxxxxx

In base 10:     22 38 61 50   26 48 N/A N/A

Base64 Output:  Wm9yaw==
```

Solution

For my solution, for the encode script, I started by making a method to convert the characters to their respective hexadecimal representations. After that I had made another method to print the numbers in binary. I stole a bit of code from my previous project. Then I took the binary from the previous statement and made a simple loop to parse them into 6 bit subdivisions and a check to fill in the extra space with x's. From this point, I used a binary string to decimal function from the previous project with the additional conditional to fill in the x's with 0's and the all 0 sections to "N/A". From here, I took the decimal representations and used them as an index to get the characters from the provided list.

For the decode script, I parsed the string and removed the = signs which were place holders for the N/A sections. I then got the index of the characters from the once again provided list and got the decimal representation. From here I converted the decimals and added them into a list where I took the 8 bit sections and disregarded what was left because it wasnt of use to me. Then I converted those sections to decimal and then used python's chr() function to convert the number to an ascii representation.

Source Code:
Base64.py (encode script)

```python
def expbintodec(binstr):
    inttot = 0
    counter = 7
    for char in binstr:
        if char == "1":
            inttot += pow(2, counter)
        counter -= 1
    return inttot


def binaryDump(stringmsg):
    dump = ""
    counter = 0
    while counter < len(stringmsg):
        placers = "0" * (8-len(bin(ord(stringmsg[counter]))[2:]))
        dump += placers
        dump += bin(ord(stringmsg[counter]))[2:]
        dump += "\t"
        counter += 1
        if counter%4 == 0:
            dump += "\n\t\t\t\t "
    return dump


def binaryDump6(stringmsg):
    dump = ""
    counter = 0
    while counter < len(stringmsg):
        placers = "0" * (8 - len(bin(ord(stringmsg[counter]))[2:]))
        dump += placers
        dump += bin(ord(stringmsg[counter]))[2:]
        counter += 1
    counter = 0
    spaceddump = ""
    fillcounter = -len(dump)
    while fillcounter < 0:
        fillcounter += 24
    while counter < len(dump):
        spaceddump += dump[counter:counter+6]
        if counter < len(dump)-6:
```

```
            spaceddump += "\t"
        counter += 6
        if counter%24 == 0:
            spaceddump += "\n\t\t\t\t "
    while fillcounter%6 > 0:
        spaceddump += "x"
        fillcounter -= 1;
    spaceddump += "\t"
    while fillcounter > 0:
        spaceddump += "xxxxxx\t"
        fillcounter -= 6;
    return spaceddump


def AsciiDump(stringmsg):
    total_string = ""
    linecounter = 0
    while(linecounter <= len(stringmsg)-1):
        iterationcounter = 0
        while iterationcounter != 4:
            total_string += stringmsg[(linecounter + iterationcounter)]
            total_string += " = $"
            total_string += (hex(ord(stringmsg[(linecounter + iterationcounter)]))[2:]).upper()
            total_string += "     "
            iterationcounter += 1
            if (linecounter + iterationcounter) > len(stringmsg)-1:
                iterationcounter = 4
        linecounter += 4
        if linecounter < len(stringmsg):
            total_string += "\n\t\t\t\t "
    return total_string


def expbintodec(binstr):
    inttot = 0
    counter = 5
    for char in binstr:
        if char == "1":
            inttot += pow(2, counter)
        counter -= 1
```

```
        return inttot


    def to_base_10(stringmsg):
        dump = ""
        counter = 0
        zero_counter = 24-(len(stringmsg)*8)%24
        nacounter = zero_counter/6
        while counter < len(stringmsg):
            placers = "0" * (8 - len(bin(ord(stringmsg[counter]))[2:]))
            dump += placers
            dump += bin(ord(stringmsg[counter]))[2:]
            counter += 1
        counter = 0
        decstr = ""
        while counter < len(dump):
            decstr += str(expbintodec(dump[counter:counter+6]))
            decstr += "\t"
            counter += 6
        while nacounter > 1:
            decstr += "N/A\t"
            nacounter -= 1
        return decstr


    baselist = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"


    def base64_string(stringmsg):
        dump = ""
        counter = 0
        zero_counter = 24-(len(stringmsg)*8)%24
        nacounter = zero_counter/6
        while counter < len(stringmsg):
            placers = "0" * (8 - len(bin(ord(stringmsg[counter]))[2:]))
            dump += placers
            dump += bin(ord(stringmsg[counter]))[2:]
            counter += 1
        counter = 0
        decstr = ""
```

```
        while counter < len(dump):
            decstr += baselist[expbintodec(dump[counter:counter+6])]
            counter += 6
        while nacounter > 1:
            decstr += "="
            nacounter -= 1
        return decstr


message = input("Enter Your Message:")
print("")
print("Original data:   " + message)
print("")
print("ASCII codes:     " + AsciiDump(message))
print("In binary:       " + binaryDump(message))
print("")
print("Groups of 8:     " + binaryDump(message))
print("")
print("Groups of 6:     " + binaryDump6(message))
print("")
print("In base 10:      " + to_base_10(message))
print("")
print("Base64 Output:   " + base64_string(message))

```

Base64toAscii.py (decode script)

```python
message = input("Enter Your Base64 message:")
baselist = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"

endmessage = message.index('=')
message = message[:endmessage]


def expbintodec(binstr):
    inttot = 0
    counter = 7
    for char in binstr:
        if char == "1":
            inttot += pow(2, counter)
        counter -= 1
    return inttot


def messagetoDec(messagestr):
    string = ""
    counter = 0
    while counter < len(messagestr):
        string += '{0:06b}'.format(baselist.index(message[counter]))
        counter += 1
    newstring = ""
    counter = 0
    while counter < int(len(string)/8):
        newstring += chr(expbintodec(string[counter*8:counter*8+8]))
        counter += 1
    return newstring


print(messagetoDec(message))
```

Tests

```
Enter Your Message:Zork

Original data:   Zork

ASCII codes:     Z = $5A     o = $6F     r = $72     k = $6B
In binary:       01011010   01101111    01110010    01101011


Groups of 8:     01011010   01101111    01110010    01101011


Groups of 6:     010110 100110  111101  110010
                 011010 11xxxx  xxxxxx  xxxxxx

In base 10:      22 38  61  50  26  48  N/A N/A

Base64 Output:   Wm9yaw==

Process finished with exit code 0
```

```
Enter Your Base64 message:Wm9yaw==
Zork

Process finished with exit code 0
```

```
Enter Your Message:I want to die

Original data:   I want to die

ASCII codes:     I = $49       = $20     w = $77     a = $61
                 n = $6E     t = $74       = $20     t = $74
                 o = $6F       = $20     d = $64     i = $69
                 e = $65
In binary:       01001001   00100000    01110111    01100001
                 01101110   01110100    00100000    01110100
                 01101111   00100000    01100100    01101001
                 01100101

Groups of 8:     01001001   00100000    01110111    01100001
                 01101110   01110100    00100000    01110100
                 01101111   00100000    01100100    01101001
                 01100101

Groups of 6:     010010 010010  000001  110111
                 011000 010110  111001  110100
                 001000 000111  010001  101111
                 001000 000110  010001  101001
                 011001 01xxxx  xxxxxx  xxxxxx

In base 10:      18 18  1   55  24  22  57  52  8   7   17  47  8   6   17  41  25  16  N/A N/A

Base64 Output:   SSB3YW50IHRvIGRpZQ==
```

```
Enter Your Base64 message:SSB3YW50IHRvIGRpZQ==
I want to die

Process finished with exit code 0
```

Conclusion

Overall, this was a fun project to code, however, I know there are a few flaws with it but I don't have enough time to correct it. For example, my decode script depends on an = sign at the end of the message or it fails. The things I did learn during this project is more about pythons formatting and built- in functionality. I look forward to the next project as I will spend more time on it.