1. Hammer

Python code: <http://www.codeskulptor.org/#user40_SSkvpa7lNirPsEp.py>

Pseudo code:

1. To start – Enter a text/letter in the input block of GUI
2. Function **word\_input** reads the text/letter given in input block.

If input -> letter then code output -> Output window/ GUI canvas block

If input -> text then code output -> Output window

1. Function **word\_input** calls function **input**
2. Function **input** takes individual letters/letter from **word\_input** of as input
3. Function **input** calls the functions **binary\_value** and **hamm**
4. Functions **binary\_value** and **hamm** take the input of function **input** after converting the individual letter/character into ASCII number using standard function **ord()**
5. Function **binary\_value**
6. First two empty lists are created
7. Then the below is executed

while( x/2 != 1):

a.append(x%2) -> append adds the value to the end of list

x /= 2

if(x%2==0):

a.append(0)

else:

a.append(1)

a.append(1)

a.reverse() -> list is reversed to the get the digits in correct order

1. To get 7 digit binary form of ASCII code below loop is executed

if(len(a) < 7): -> len gives the length of the list/string

y = len(a)

z = 7 - y

for i in range(0, z):

b.append(0)

for j in range(0, y):

b.append(a[j])

else:

b = a

1. End the function returns b -> 7 digit binary representation of ASCII code

1. Function **hamm**
2. An empty list **y** and a list **c** with 11 zeroes are created.
3. Similarly some lists as created as follows.

three = [0, 0, 0, 0]

five = [0, 0, 0, 0]

six = [0, 0, 0, 0]

seven = [0, 0, 0, 0]

nine = [0, 0, 0, 0]

ten = [0, 0, 0, 0]

eleven = [0, 0, 0, 0]

1. Then the following code is implemented.

x = ord(t)

y = binary\_value(x)

c[2] = y[0]

c[4] = y[1]

c[5] = y[2]

c[6] = y[3]

c[8] = y[4]

c[9] = y[5]

c[10] = y[6]

if(c[2]==1):

three[0] = three[1] =1

if(c[4]==1):

five[0] = five[2] = 1

if(c[5]==1):

six[1] = six[2] = 1

if(c[6]==1):

seven[0] = seven[1] = seven[2] = 1

if(c[8]==1):

nine[0] = nine[3] = 1

if(c[9]==1):

ten[1] = ten[3] = 1

if(c[10]==1):

eleven[0] = eleven[1] = eleven[3] = 1

ze = three[0]+five[0]+six[0]+seven[0]+nine[0]+ten[0]+eleven[0]

on = three[1]+five[1]+six[1]+seven[1]+nine[1]+ten[1]+eleven[1]

tw = three[2]+five[2]+six[2]+seven[2]+nine[2]+ten[2]+eleven[2]

th = three[3]+five[3]+six[3]+seven[3]+nine[3]+ten[3]+eleven[3]

if(ze%2 == 1):

c[0] = ze%2

if(on%2 == 1):

c[1] = on%2

if(tw%2 == 1):

c[3] = tw%2

if(th%2 == 1):

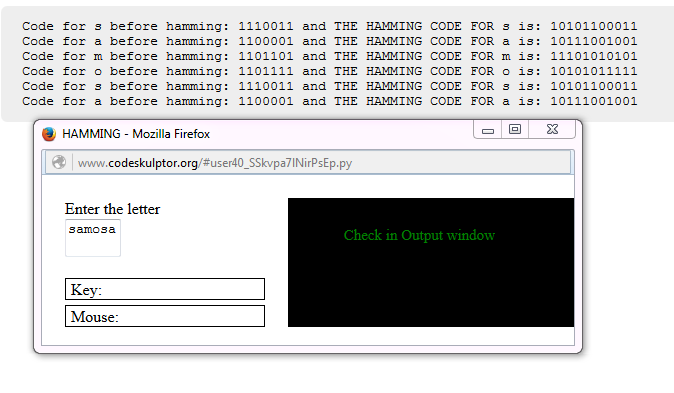
c[7] = th%2

return c

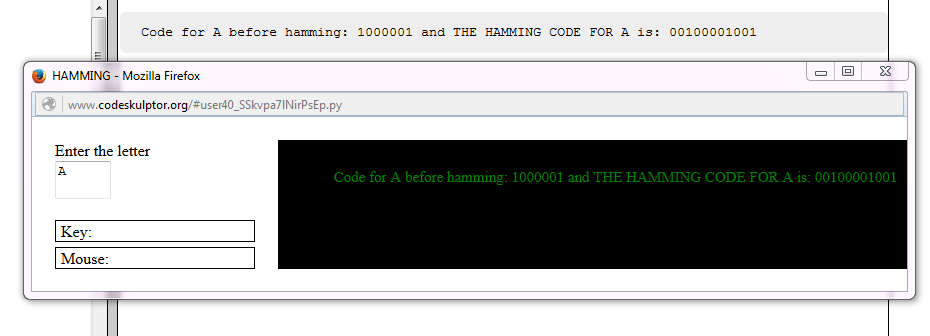
1. End list **c** is returned which is 11 digit binary code representing the hamming code.

OUTPUT SNAPSHOTS FOR HAMMER

For input given as “samosa”



For input given as “A”



1. De-hammer

Python code: <http://www.codeskulptor.org/#user40_vCCeBzHwPoAOkdg.py>

Pseudo code

1. Start – enter the code for letter to be de-hammed

To get text/word as output enter the next hamming code after the deleting the initial.

1. Function **code\_input** takes the input and converts it to string using standard function **str()** which is initialized to s1
2. Numerical value of ASCII code is calculated using the following

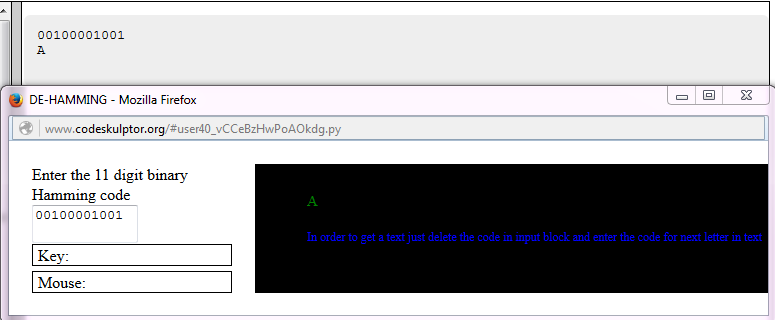
64\*(int(s1[2])) + 32\*(int(s1[4])) + 16\*(int(s1[5])) + 8\*(int(s1[6])) + 4\*(int(s1[8])) + 2\*(int(s1[9])) + (int(s1[10]))

Where standard function **int()** gives the integer value

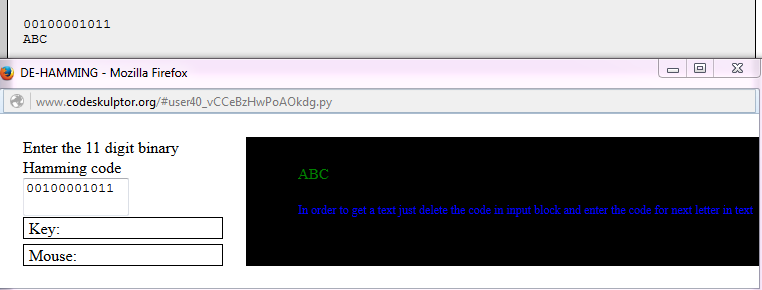
1. Next character/alphabet with the obtained ASCII code is calculated using the standard function **chr()**
2. Function **code\_input** contains string concatenation to form a word/text if multiple hamming codes are given in input.

SNAPSHOTS FOR DEHAMMING

Hamming codes for ‘A’, ‘B’ & ‘C’ are given as input.







1. Error Correction

Python code: <http://www.codeskulptor.org/#user40_b5WzQtQqmBJ6hdR.py>

Pseudo code:

1. Start – Enter the 11 digit hamming code which needed to be corrected
2. Function code\_input takes the input and converts it into string using the standard function **str()** and initialized to s1.
3. Next the following are initialized

counter = 0

one = [0, 0, 0, 0]

two = [0, 0, 0, 0]

three = [0, 0, 0, 0]

four = [0, 0, 0, 0]

five = [0, 0, 0, 0]

six = [0, 0, 0, 0]

seven = [0, 0, 0, 0]

eight = [0, 0, 0, 0]

nine = [0, 0, 0, 0]

ten = [0, 0, 0, 0]

eleven = [0, 0, 0, 0]

1. String is changed with respect to the following statements

if(int(s1[0])==1):

one[0] = 1

if(int(s1[1])==1):

two[1] = 1

if(int(s1[2])==1):

three[0] = three[1] = 1

if(int(s1[3])==1):

four[2] = 1

if(int(s1[4])==1):

five[0] = five[2] = 1

if(int(s1[5])==1):

six[1] = six[2] = 1

if(int(s1[6])==1):

seven[0] = seven[1] = seven[2] = 1

if(int(s1[7])==1):

eight[3] = 1

if(int(s1[8])==1):

nine[0] = nine[3] = 1

if(int(s1[9])==1):

ten[1] = ten[3] = 1

if(int(s1[10])==1):

eleven[0] = eleven[1] = eleven[3] = 1

1. Next sum of elements with index ‘0’ in the above lists are assigned to integer ‘a1’.

Similarly sum of elements with ‘1’ assigned to ‘a2’

With ‘2’ assigned to ‘a4’

And with ‘3’ assigned to ‘a8’

1. Variable counter is updated with respect to the following statements

if(a1%2==1):

counter += 1

if(a2%2==1):

counter += 2

if(a4%2==1):

counter += 4

if(a8%2==1):

counter += 8

1. String s1 is updated at the position **counter – 1** using string concatenation using the following code

if(int(s1[counter-1])==1):

s2 = s1[0: counter-1] + '0' + s1[counter:] -> s1 is updated to s2

elif(int(s1[counter-1])==1):

s2 = s1[0: counter-1] + '1' + s1[counter:]

elif(counter==0):

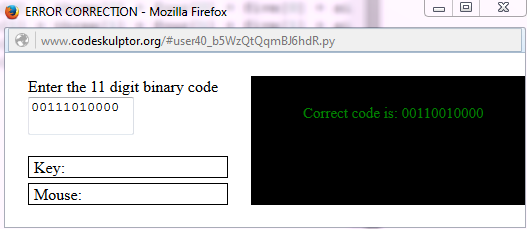
s2 = s1

1. End - Next the two strings s1 and s2 are compared and the appropriate result is displayed in GUI.

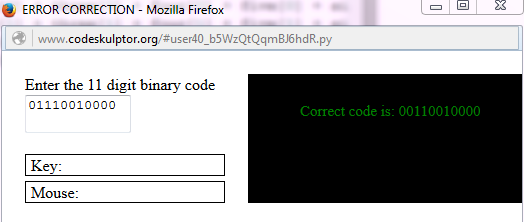
SNAPSHOTS FOR ERROR CORRECTION

Correct hamming code for ‘H’ = 00110010000

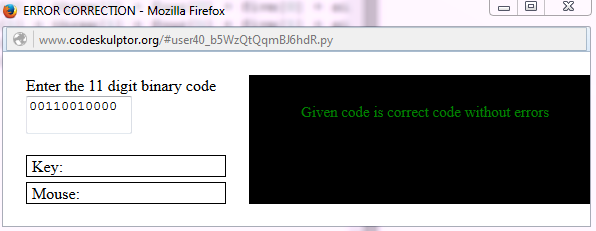
Case 1: “00111010000” (Change is made at 5th position) is given as input



Case 2:“01110010000” (Change is made at 2nd position) is given as input



Case3:“00110010000” – Original code is given as input

****