

A T
C G

Mini-project

Encoding
Information into
DNA bases

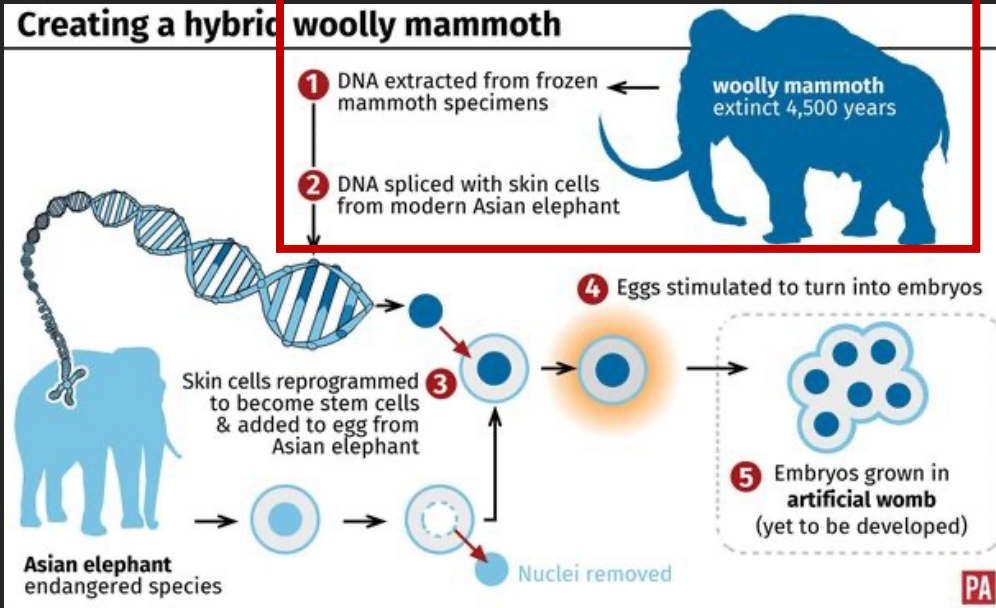
Presented by @SujirapornPak
August 4th, 2020

2. Huge Capacity data storage

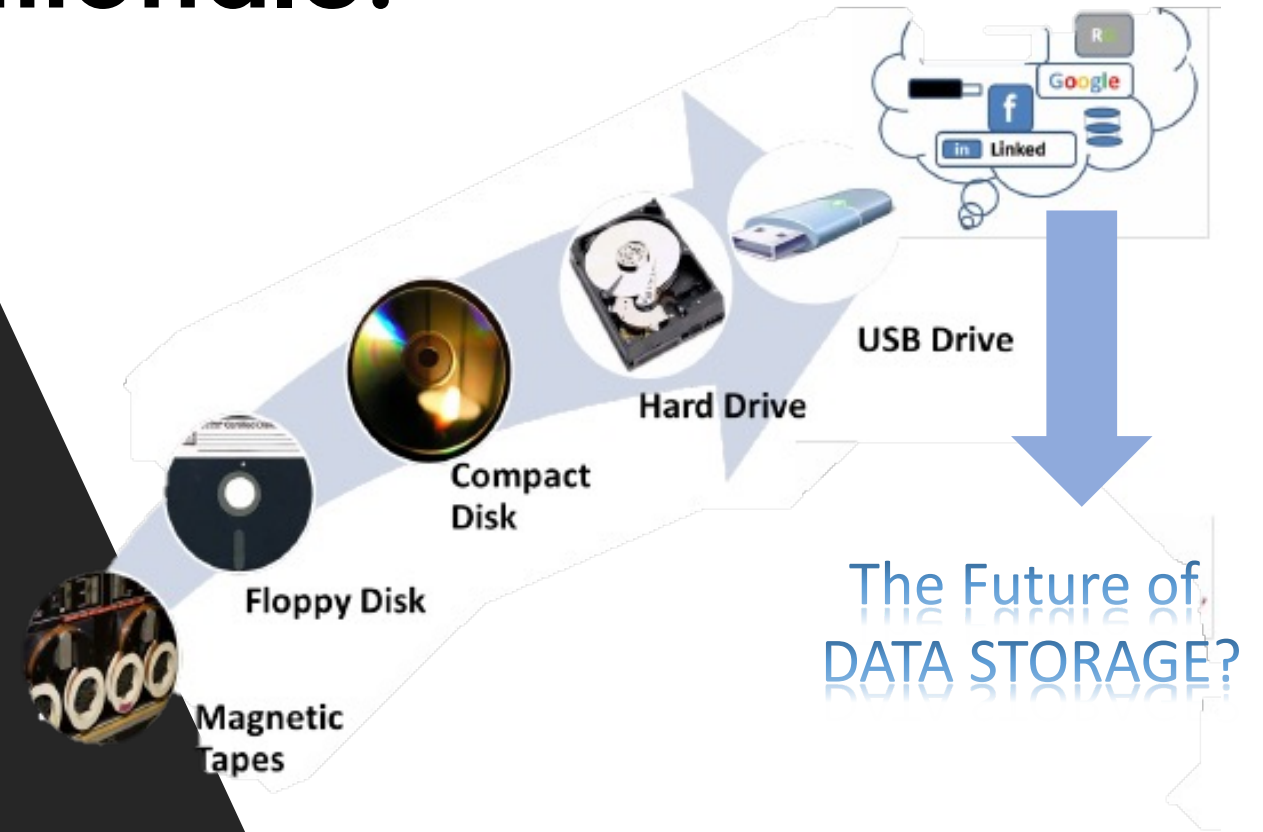
>>> 1 g. of DNA is capable of holding 700 terabytes (10^{12}) of data!
(close to 1 petabyte (10^{15}))

3. Durable

Creating a hybrid woolly mammoth



Rationale:








1. Storage Limits

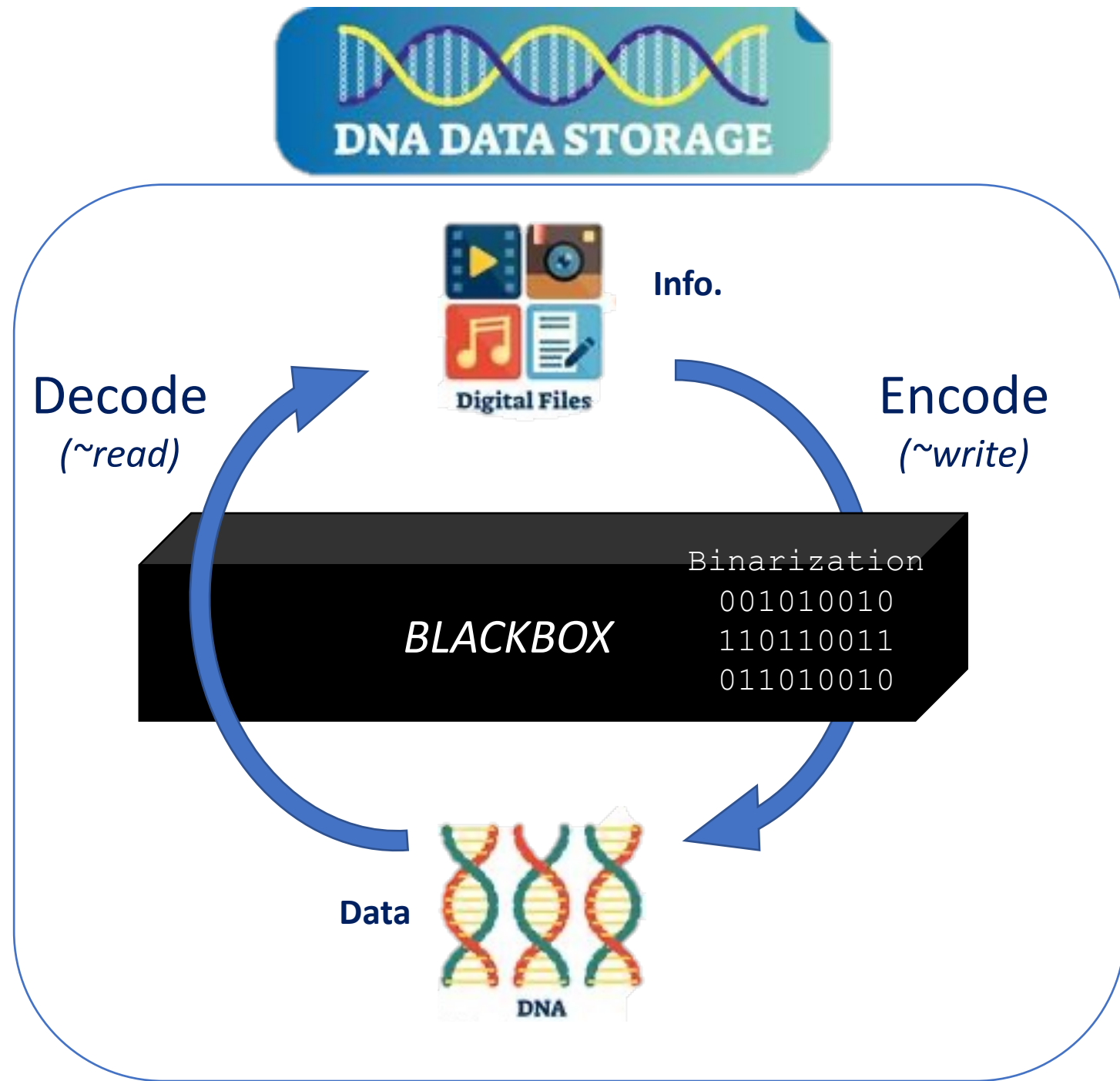
Harvard cloning scientist is using DNA from a 42,000-year old carcass to see the Ice Age species rise from the dead!

STORAGE LIMITS


Estimates based on bacterial genetics suggest that digital DNA could one day rival or exceed today's storage technology.

	 Hard disk	 Flash memory	 Bacterial DNA	WEIGHT OF DNA NEEDED TO STORE WORLD'S DATA
Read-write speed (μ s per bit)	$\sim 3,000\text{--}5,000$	~ 100	< 100	  $\sim 1\text{ kg}$ ©nature
Data retention (years)	> 10	> 10	> 100	
Power usage (watts per gigabyte)	~ 0.04	$\sim 0.01\text{--}0.04$	$< 10^{-10}$	
Data density (bits per cm^3)	$\sim 10^{13}$	$\sim 10^{16}$	$\sim 10^{19}$	

Life Cycle?



How to convert the text "hope" into binary (8-bits)?



Characters:	h	o	p	e
ASCII Values:	104	111	112	101
Binary Values:	01101000	01101111	01110000	01100101
Bits:	8	8	8	8

ASCII (ASS-kee) abbreviated from **American Standard Code for Information Interchange**.

- **ASCII** is a character encoding standard for electronic communication.
- **ASCII** codes represent text in computers and other devices.

1 byte (= 8 bits)

can store one character,
e.g. 'A' , '1' , '\$' , '+'

Character Encoding

Decimal - Binary - Octal - Hex – ASCII Conversion Chart

2⁸ = 256 char.

>>> Here, 96
characters are
enough!

Idea!

Make Dictionary

Key <-> Key-Values

Binary <-> ASCII

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	\$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

How many patterns with N Bits?

	Number of bits	Different Patterns
$(\log_2 2^1)$	1	0 1
$(\log_2 2^2)$	2	00 01 10 11
$(\log_2 2^3)$	3	000 001 010 011 100 101 110 111



4 patterns which also fits to
4 characters of DNA bases

For ex 00 > A
01 > T
10 > C
11 > G

Step 1: Convert Characters to Binary Values and encode to DNA string

- Make a Dictionary of 96 characters and 8-bits binary string (eg. 00010111) as Keys and Key-values, respectively.
- Get input words/sentence from user and return the binary string and DNA sequence in the .txt file.

Step 2: Convert DNA sequence to Binary Values and decode to Characters/words

- Make a Dictionary of Reciprocal Pairs of those in step 1 as Keys and Key-values.
- Get input DNA sequence from user and return the decoded DNA in binary and characters in .txt file.

Final Exam

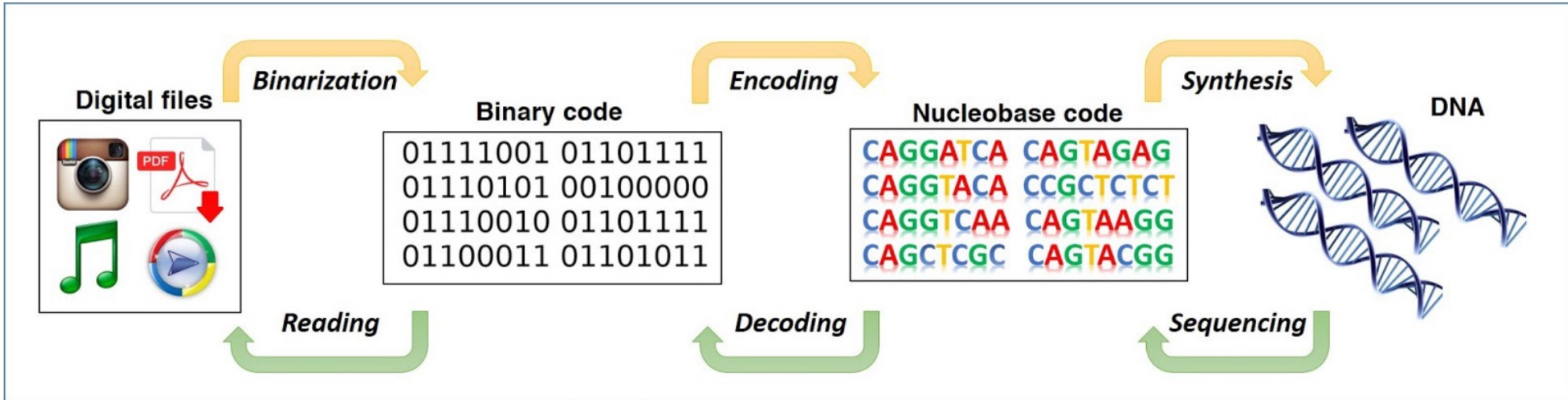
Project: Encoding information into DNA bases.

Methods:

1. Convert characters to binary values and encode to DNA string.
2. Convert DNA sequence to binary values and decode to characters or words.

```
In [3]: from IPython.display import Image  
Image("Concept-DNAstorage.png")
```

Out[3]:



```

In [45]: def encod_WordstoDNA(yy):
yy = input('Your input words: ')
yy = list(yy)

## Note: This dictionary contains only 96 characters, while 8-bits can encode 2^8 = 256 possibility of character.
ASCII = {' ': '00100000', '!': '00100001', '"': '00100010', '#': '00100011',
'$': '00100100', '%': '00100101', '&': '00100110', "'": '00100111',
'(': '00101000', ')': '00101001', '*': '00101010', '+': '00101011',
',': '00101100', '-': '00101101', '.': '00101110', '/': '00101111',
'0': '00110000', '1': '00110001', '2': '00110010', '3': '00110011',
'4': '00110100', '5': '00110101', '6': '00110110', '7': '00110111',
'8': '00111000', '9': '00111001', ':': '00111010', ';': '00111011',
'<': '00111100', '=': '00111101', '>': '00111110', '?': '00111111',
'@': '01000000', 'A': '01000001', 'B': '01000010', 'C': '01000011',
'D': '01000100', 'E': '01000101', 'F': '01000110', 'G': '01000111',
'H': '01001000', 'I': '01001001', 'J': '01001010', 'K': '01001011',
'L': '01001100', 'M': '01001101', 'N': '01001110', 'O': '01001111',
'P': '01010000', 'Q': '01010001', 'R': '01010010', 'S': '01010011',
'T': '01010100', 'U': '01010101', 'V': '01010110', 'W': '01010111',
'X': '01011000', 'Y': '01011001', 'Z': '01011010', '[': '01011011',
']': '01011101', '^': '01011110', '_': '01011111', '\': '01011100',
`': '01100000', 'a': '01100001', 'b': '01100010', 'c': '01100011',
'd': '01100100', 'e': '01100101', 'f': '01100110', 'g': '01100111',
'h': '01101000', 'i': '01101001', 'j': '01101010', 'k': '01101011',
'l': '01101100', 'm': '01101101', 'n': '01101110', 'o': '01101111',
'p': '01110000', 'q': '01110001', 'r': '01110010', 's': '01110011',
't': '01110100', 'u': '01110101', 'v': '01110110', 'w': '01110111',
'x': '01111000', 'y': '01111001', 'z': '01111010', '{': '01111011',
'|': '01111100', '}' : '01111101', '~': '01111110', 'DEL': '01111111'}

## Step1: To convert words into long binary string.
bina = ''
for i in list(range(len(yy))):
    char = yy[i]
    if char in ASCII:
        bina += ASCII[char]

```

```

## Step2: To encode 2 bits to DNA base.
simbasedict = {'00':'A', '01':'T', '10':'C', '11':'G'}
myencode = ''
for i in list(range(0,len(bina),2)):
    bits = bina[i:i+2]
    if bits in simbasedict:
        myencode += simbasedict[bits]
GCcnt = (myencode.count('G')+myencode.count('C'))/len(myencode)*100
ATcnt = (myencode.count('A')+myencode.count('T'))/len(myencode)*100

## Step3: To split long binary string into disered length (window size).
binar = ''
for ii in list(range(0,len(bina),20)):
    line = bina[ii:ii+20]
    binar += line+('\n')

## Step4: To split long DNA into disered length (window size).
window = ''
for ii in list(range(0,len(myencode),20)):
    line2 = myencode[ii:ii+20]
    window += line2+('\n')

## Step5: To write output into new text file.
stri = ''
f = open('encode_Words_to_DNA.txt','w')
f.write('Your input words:'+'\n'+stri.join(yy)+'\n'+'\n')
f.write('Words encoded to binary, '+str(len(bina))+ ' bits in total, based on the ASCII format are shown below:\n')
f.write(binar+'\n')
f.write('Binary sequence encoded to DNA sequence is shown below:\n')
f.write(window+'\n')
f.write('The length of this DNA sequence is '+str(len(myencode))+ ' bp. GC and AT content are '+str(GCcnt)+'% and')
f.close()

## Step6: To print both outputs to user presented in jupyter nb..
print ('Your words can be converted into binary code,',len(bina),'bits in total, based on the ASCII format as:\n')
print ('Binary sequence encoded into DNA sequence is shown below:\n'+window)
print ('The length of this DNA sequence is',len(myencode),'bp. GC and AT content are',GCcnt,'% and',ATcnt,'% , re')
print ('Your output file is saved in the text file as "encode_Words_to_DNA.txt".')

```

```
1 yy = 'hello'
2 encod_WordstoDNA(yy)
```

Your input words: Wonderful

Your words can be converted into binary code, 72 bits in total, based on the ASCII format as:

```
01010111011011110110
11100110010001100101
01110010011001100111
010101101100
```

Binary sequence encoded into DNA sequence is shown below:

```
TTTGTCGGTCGCTCTATCTT
TGACTCTCTGTTTCGA
```

The length of this DNA sequence is 36 bp. GC and AT content are 44.44444444444444 % and 55.55555555555556 %, respectively.

Your output file is saved in the text file as "encode_Words_to_DNA.txt".


```

In [48]: def decode_DNAtoWords(xx):
xx = input('Your input DNA sequence: ')
xx = xx.replace(' ', '')
GCcnt = (xx.count('G')+xx.count('C'))/len(xx)*100
ATcnt = (xx.count('A')+xx.count('T'))/len(xx)*100

## Step1: To convert DNA into long binary string.
simbasedict2 = {'A':'00', 'T':'01', 'C':'10', 'G':'11'}
mydecode = ''
for i in list(range(0, len(xx))):
    base = xx[i:i+1]
    if base in simbasedict2:
        mydecode += simbasedict2[base]

## Step2: To decode 8-bits to characters.
## Noted that 8-bits results in 2^8 = 256 possibility of characters.
## However, this dictionary contains only 96 characters, which might cause 'KeyError' if any 8-bits don't exist.
ASCIIex = {'00100000': ' ', '00100001': '!', '00100010': '"', '00100011': '#',
'00100100': '$', '00100101': '%', '00100110': '&', '00100111': "'",
'00101000': '(', '00101001': ')', '00101010': '*', '00101011': '+',
'00101100': ',', '00101101': '-', '00101110': '.', '00101111': '/',
'00110000': '0', '00110001': '1', '00110010': '2', '00110011': '3',
'00110100': '4', '00110101': '5', '00110110': '6', '00110111': '7',
'00111000': '8', '00111001': '9', '00111010': ':', '00111011': ';',
'00111100': '<', '00111101': '=', '00111110': '>', '00111111': '?',
'01000000': '@', '01000001': 'A', '01000010': 'B', '01000011': 'C',
'01000100': 'D', '01000101': 'E', '01000110': 'F', '01000111': 'G',
'01001000': 'H', '01001001': 'I', '01001010': 'J', '01001011': 'K',
'01001100': 'L', '01001101': 'M', '01001110': 'N', '01001111': 'O',
'01010000': 'P', '01010001': 'Q', '01010010': 'R', '01010011': 'S',
'01010100': 'T', '01010101': 'U', '01010110': 'V', '01010111': 'W',
'01011000': 'X', '01011001': 'Y', '01011010': 'Z', '01011011': '[',
'01011101': ']', '01011110': '^', '01011111': '_', '01011100': '"\'"',
'01100000': '`', '01100001': 'a', '01100010': 'b', '01100011': 'c',
'01100100': 'd', '01100101': 'e', '01100110': 'f', '01100111': 'g',
'01101000': 'h', '01101001': 'i', '01101010': 'j', '01101011': 'k',
'01101100': 'l', '01101101': 'm', '01101110': 'n', '01101111': 'o',
'01110000': 'p', '01110001': 'q', '01110010': 'r', '01110011': 's',

```

Step2: To decode 8-bits to characters.

Noted that 8-bits results in $2^8 = 256$ possibility of characters.

However, this dictionary contains only 96 characters, which might cause 'KeyError' if any 8-bits don't exist.

```
ASCIIex = {'00100000': ' ', '00100001': '!', '00100010': '"', '00100011': '#',  
          '00100100': '$', '00100101': '%', '00100110': '&', '00100111': "'",  
          '00101000': '(', '00101001': ')', '00101010': '*', '00101011': '+',  
          '00101100': ',', '00101101': '-', '00101110': '.', '00101111': '/',  
          '00110000': '0', '00110001': '1', '00110010': '2', '00110011': '3',  
          '00110100': '4', '00110101': '5', '00110110': '6', '00110111': '7',  
          '00111000': '8', '00111001': '9', '00111010': ':', '00111011': ';',  
          '00111100': '<', '00111101': '=', '00111110': '>', '00111111': '?',  
          '01000000': '@', '01000001': 'A', '01000010': 'B', '01000011': 'C',  
          '01000100': 'D', '01000101': 'E', '01000110': 'F', '01000111': 'G',  
          '01001000': 'H', '01001001': 'I', '01001010': 'J', '01001011': 'K',  
          '01001100': 'L', '01001101': 'M', '01001110': 'N', '01001111': 'O',  
          '01010000': 'P', '01010001': 'Q', '01010010': 'R', '01010011': 'S',  
          '01010100': 'T', '01010101': 'U', '01010110': 'V', '01010111': 'W',  
          '01011000': 'X', '01011001': 'Y', '01011010': 'Z', '01011011': '[',  
          '01011101': ']', '01011110': '^', '01011111': '_', '01011100': '"',  
          '01100000': '`', '01100001': 'a', '01100010': 'b', '01100011': 'c',  
          '01100100': 'd', '01100101': 'e', '01100110': 'f', '01100111': 'g',  
          '01101000': 'h', '01101001': 'i', '01101010': 'j', '01101011': 'k',  
          '01101100': 'l', '01101101': 'm', '01101110': 'n', '01101111': 'o',  
          '01110000': 'p', '01110001': 'q', '01110010': 'r', '01110011': 's',  
          '01110100': 't', '01110101': 'u', '01110110': 'v', '01110111': 'w',  
          '01111000': 'x', '01111001': 'y', '01111010': 'z', '01111011': '{',  
          '01111100': '|', '01111101': '}', '01111110': '~', '01111111': 'DEL'}
```

```
charac = ''
```

```
for i in list(range(0, len(mydecode), 8)):
```

```
    byte = mydecode[i:i+8]
```

```
    try:
```

```
        ASCIIex[byte]
```

```
    except KeyError:
```

```
        charac += '#'
```

```
    else:
```

```
        charac += ASCIIex[byte]
```


Step3: To split long binary string into disered length (window size).

```
bytee = ''  
for ii in list(range(0,len(mydecode),20)):  
    line = mydecode[ii:ii+20]  
    bytee += line+'\n'
```

Step4: To write output into new text file.

```
f = open('decode_DNA_to_Words.txt','w')  
f.write('Your input DNA sequence:'+'\n'+xx+'\n')  
f.write('The length of this DNA sequence is '+str(len(xx))+' bp. GC and AT content are '+str(GCcnt)+'% and '+str(ATcnt)+'%', respectively)  
f.write('DNA sequence decoded to binary, '+str(len(mydecode))+' bits in total, is shown below:\n')  
f.write(bytee+'\n')  
f.write('Binary sequence decoded to characters based on the ASCII format is shown below:\n')  
f.write(charac+'\n')  
f.close()
```

Step5: To print both outputs to user presented in jupyter nb.

```
print ('The length of this DNA sequence is',len(xx),'bp. GC and AT content are',GCcnt,'% and ',ATcnt,'% , respectively')  
print ('DNA sequence decoded to binary,',(len(mydecode)),'bits in total, is shown below:\n'+bytee)  
print ('Binary sequence decoded to characters based on the ASCII format is shown below:\n'+charac+'\n')  
print ('Your output file is saved in the text file as "decode_DNA_to_Words.txt".')
```

```
1 xx = 'hello'
2 decode_DNAtoWords(xx)
```

Your input DNA sequence: TTTGTCGGTCGCTCTATCTTTGACTCTCTGTTTCGA

The length of this DNA sequence is 36 bp. GC and AT content are 44.44444444444444 % and 55.55555555555556 %, respectively.

DNA sequence decoded to binary, 72 bits in total, is shown below:

```
01010111011011110110
11100110010001100101
01110010011001100111
010101101100
```

Binary sequence decoded to characters based on the ASCII format is shown below:

Wonderful

Your output file is saved in the text file as "decode_DNA_to_Words.txt".