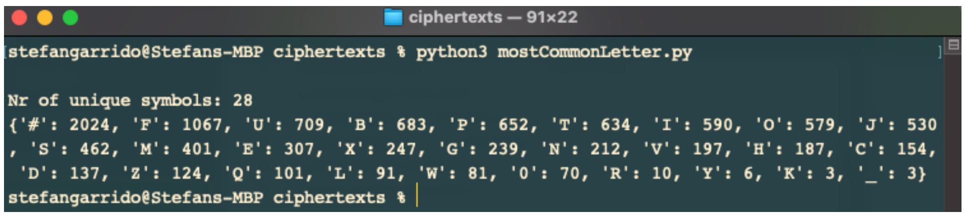
DD2520, Applied Cryptography
Cryptanalysis of Ciphertexts

Stefan Garrido: sagv@kth.se

January 26, 2023

Ciphertext 1

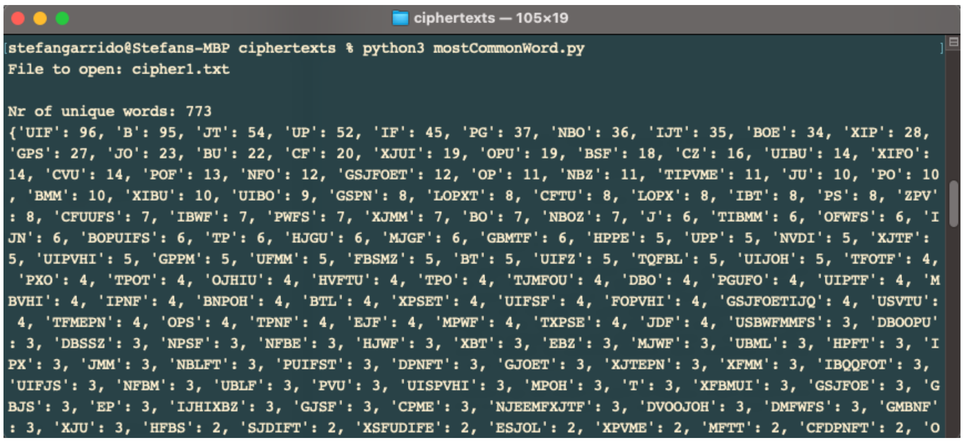
The analysis began with an initial overview of the cipher to see if any patterns were observable. The first thing to remark was the large amount of occurrences of the character #. Additionally, "words" in between # seemed to occur more than once. A quick comparison to ciphertext 2 showed the frequency of the character # was minimal in the second ciphertext. A python program was created to measure the frequency of each symbol in ciphertext 1, the results are seen on figure 1. Based on this, an assumption was made that the character # could potentially be used to substitute the underscore character, which corresponds to an empty/space/blank character.



```
stefangarrido@Stefans-MBP ciphertexts % python3 mostCommonLetter.py

Nr of unique symbols: 28
{'#': 2024, 'F': 1067, 'U': 709, 'B': 683, 'P': 652, 'T': 634, 'I': 590, 'O': 579, 'J': 530
, 'S': 462, 'M': 401, 'E': 307, 'X': 247, 'G': 239, 'N': 212, 'V': 197, 'H': 187, 'C': 154,
 'D': 137, 'Z': 124, 'Q': 101, 'L': 91, 'W': 81, '0': 70, 'R': 10, 'Y': 6, 'K': 3, '_': 3}
stefangarrido@Stefans-MBP ciphertexts %
```

Figure 1: Output of the program *mostCommonLetter.py*

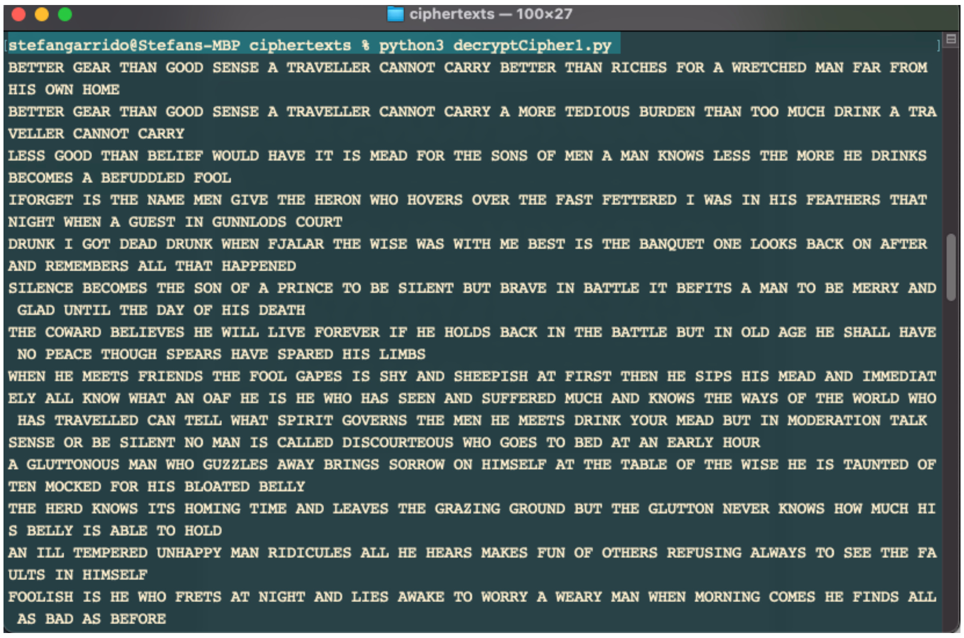
A python program was created to analyse the frequency of each encrypted word found in between #, see figure 2 for a partial result of the output.
Based on the analysis of the Oxford English Corpus and the Corpus of Contemporary American English (COCA) list [1], the most common word in English is the word "**the**". The encrypted word with most occurrences, according to the python program, was "**UIF**". An assumption was made that **UIF** corresponds to the word **the**, and as such: U=T, I=H, and F=E. The mapping from F to E seems to match the frequency of the letter E seen on figure 1. Based on common English words, the same type of assumptions were made for the words "B" and "UP", resulting in: B=A, U=T, and P=O.


ciphertexts — 105×19

```
stefangarrido@Stefans-MBP ciphertexts % python3 mostCommonWord.py
File to open: cipher1.txt

Nr of unique words: 773
{'UIF': 96, 'B': 95, 'JT': 54, 'UP': 52, 'IF': 45, 'PG': 37, 'NBO': 36, 'IJT': 35, 'BOE': 34, 'XIP': 28,
'GPS': 27, 'JO': 23, 'BU': 22, 'CF': 20, 'XJUI': 19, 'OPU': 19, 'BSF': 18, 'CZ': 16, 'UIBU': 14, 'XIFO':
14, 'CVU': 14, 'POF': 13, 'NFO': 12, 'GSJFOET': 12, 'OP': 11, 'NBZ': 11, 'TIPVME': 11, 'JU': 10, 'PO': 10
, 'BMM': 10, 'XIBU': 10, 'UIBO': 9, 'GSPN': 8, 'LOPXT': 8, 'CFTU': 8, 'LOPX': 8, 'IBT': 8, 'PS': 8, 'ZPV'
: 8, 'CFUUFS': 7, 'IBWF': 7, 'PWFS': 7, 'XJMM': 7, 'BO': 7, 'NBOZ': 7, 'J': 6, 'TIBMM': 6, 'OFWFS': 6, 'I
JN': 6, 'BOPUIFS': 6, 'TP': 6, 'HJGU': 6, 'MJGF': 6, 'GBMTF': 6, 'HPPE': 5, 'UPP': 5, 'NVDI': 5, 'XJTF':
5, 'UIPVHI': 5, 'GPPM': 5, 'UFMM': 5, 'FBSMZ': 5, 'BT': 5, 'UIFZ': 5, 'TQFBL': 5, 'UIJOH': 5, 'TFOTF': 4,
 'PXO': 4, 'TPOT': 4, 'OJHIU': 4, 'HVFTU': 4, 'TPO': 4, 'TJMFOU': 4, 'DBO': 4, 'PGUFO': 4, 'UIPTF': 4, 'M
BVHI': 4, 'IPNF': 4, 'BNPOH': 4, 'BTL': 4, 'XPSET': 4, 'UIFSF': 4, 'FOPVHI': 4, 'GSJFOETIJQ': 4, 'USVTU':
 4, 'TFMEPN': 4, 'OPS': 4, 'TPNF': 4, 'EJF': 4, 'MPWF': 4, 'TXPSE': 4, 'JDF': 4, 'USBWFMMFS': 3, 'DBOOPU'
: 3, 'DBSSZ': 3, 'NPSF': 3, 'NFBE': 3, 'HJWF': 3, 'XBT': 3, 'EBZ': 3, 'MJWF': 3, 'UBML': 3, 'HPFT': 3, 'I
PX': 3, 'JMM': 3, 'NBLFT': 3, 'PUIFST': 3, 'DPNFT': 3, 'GJOET': 3, 'XJTEPN': 3, 'XFMM': 3, 'IBQQFOT': 3,
'UIFJS': 3, 'NFBM': 3, 'UBLF': 3, 'PVU': 3, 'UISPVHI': 3, 'MPOH': 3, 'T': 3, 'XFBMUI': 3, 'GSJFOE': 3, 'G
BJS': 3, 'EP': 3, 'IJHIXBZ': 3, 'GJSF': 3, 'CPME': 3, 'NJEEMFXJTF': 3, 'DVOOJOH': 3, 'DMFWFS': 3, 'GMBNF'
: 3, 'XJU': 3, 'HFBS': 2, 'SJDIFT': 2, 'XSFUDIFE': 2, 'ESJOL': 2, 'XPVME': 2, 'MFTT': 2, 'CFDPNFT': 2, 'O
```

Figure 2: Partial output of the program *mostCommonWord.py*

As a results of these assumptions and the given hint in the assignment it was concluded that a Caesar cipher was used, where each letter was encrypted by shifting one step to the right. A third python program was created to reverse the encryption. The results seen on figure 3 shows that ciphertext 1 was successfully decoded.

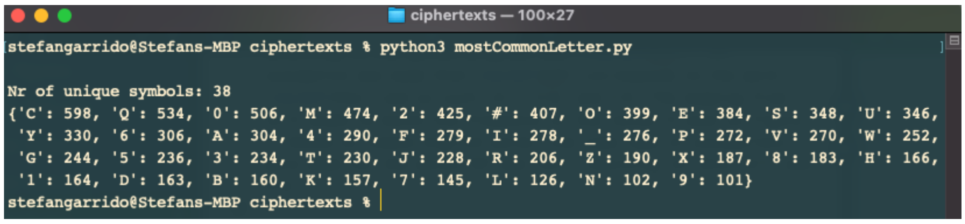

ciphertexts — 100×27

```
stefangarrido@Stefans-MBP ciphertexts % python3 decryptCipher1.py
BETTER GEAR THAN GOOD SENSE A TRAVELLER CANNOT CARRY BETTER THAN RICHES FOR A WRETCHED MAN FAR FROM
HIS OWN HOME
BETTER GEAR THAN GOOD SENSE A TRAVELLER CANNOT CARRY A MORE TEDIOUS BURDEN THAN TOO MUCH DRINK A TRA
VELLER CANNOT CARRY
LESS GOOD THAN BELIEF WOULD HAVE IT IS MEAD FOR THE SONS OF MEN A MAN KNOWS LESS THE MORE HE DRINKS
BECOMES A BEFUDDLED FOOL
IFORGET IS THE NAME MEN GIVE THE HERON WHO HOVERS OVER THE FAST FETTERED I WAS IN HIS FEATHERS THAT
NIGHT WHEN A GUEST IN GUNNLODS COURT
DRUNK I GOT DEAD DRUNK WHEN FJALAR THE WISE WAS WITH ME BEST IS THE BANQUET ONE LOOKS BACK ON AFTER
AND REMEMBERS ALL THAT HAPPENED
SILENCE BECOMES THE SON OF A PRINCE TO BE SILENT BUT BRAVE IN BATTLE IT BEFITS A MAN TO BE MERRY AND
 GLAD UNTIL THE DAY OF HIS DEATH
THE COWARD BELIEVES HE WILL LIVE FOREVER IF HE HOLDS BACK IN THE BATTLE BUT IN OLD AGE HE SHALL HAVE
 NO PEACE THOUGH SPEARS HAVE SPARED HIS LIMBS
WHEN HE MEETS FRIENDS THE FOOL GAPES IS SHY AND SHEEPISH AT FIRST THEN HE SIPS HIS MEAD AND IMMEDIAT
ELY ALL KNOW WHAT AN OAF HE IS HE WHO HAS SEEN AND SUFFERED MUCH AND KNOWS THE WAYS OF THE WORLD WHO
 HAS TRAVELLED CAN TELL WHAT SPIRIT GOVERNS THE MEN HE MEETS DRINK YOUR MEAD BUT IN MODERATION TALK
SENSE OR BE SILENT NO MAN IS CALLED DISCOURTEOUS WHO GOES TO BED AT AN EARLY HOUR
A GLUTTONOUS MAN WHO GUZZLES AWAY BRINGS SORROW ON HIMSELF AT THE TABLE OF THE WISE HE IS TAUNTED OF
TEN MOCKED FOR HIS BLOATED BELLY
THE HERD KNOWS ITS HOMING TIME AND LEAVES THE GRAZING GROUND BUT THE GLUTTON NEVER KNOWS HOW MUCH HI
S BELLY IS ABLE TO HOLD
AN ILL TEMPERED UNHAPPY MAN RIDICULES ALL HE HEARS MAKES FUN OF OTHERS REFUSING ALWAYS TO SEE THE FA
ULTS IN HIMSELF
FOOLISH IS HE WHO FRETS AT NIGHT AND LIES AWAKE TO WORRY A WEARY MAN WHEN MORNING COMES HE FINDS ALL
 AS BAD AS BEFORE
```

Figure 3: Partial output of the program *decryptCipher1.py*

Ciphertext 2

The analysis began with an initial overview of the cipher to see if any patterns were observable. Unlike ciphertext 1, no obvious patterns could be observed. A frequency analysis was done on ciphertext 2 using the same program used in ciphertext 1. The results seen on figure 4 shows that the frequency of the letters seems to be more uniformly distributed compared to ciphertext 1.



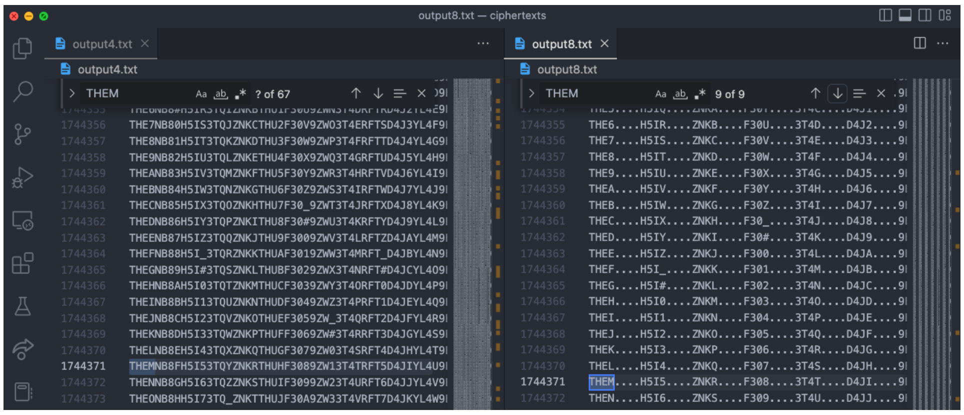
```
stefangarrido@Stefans-MBP ciphertexts % python3 mostCommonLetter.py

Nr of unique symbols: 38
{'C': 598, 'Q': 534, '0': 506, 'M': 474, '2': 425, '#': 407, 'O': 399, 'E': 384, 'S': 348, 'U': 346,
 'Y': 330, '6': 306, 'A': 304, '4': 290, 'F': 279, 'I': 278, '_': 276, 'P': 272, 'V': 270, 'W': 252,
 'G': 244, '5': 236, '3': 234, 'T': 230, 'J': 228, 'R': 206, 'Z': 190, 'X': 187, '8': 183, 'H': 166,
 '1': 164, 'D': 163, 'B': 160, 'K': 157, '7': 145, 'L': 126, 'N': 102, '9': 101}
stefangarrido@Stefans-MBP ciphertexts %
```

Figure 4: Partial output of the program *mostCommonLetter.py*

This result seemed to hint to the use of a Vigénère cipher, which is an extended Caesar cipher that uses multiple keys. The Vigénère key was assumed to have the format: k = $(k_0,...,k_{l-1})$, where $k_i \in Z_{38}$. Given this assumption, a python program was created in order to brute force the combination of keys. The program alternates k_i for every position based on the given key length and tries every combination of the given alphabet. The idea was to start with a key length of two, analyse the results, and increase the key length if needed. Only the first line of the ciphertext 2 was used in order to keep the brute forcing to a reasonable time and allow for reasonable amount of output. Additionally, in order to minimize the amount of brute forcing, an assumption was made that the first symbol in the ciphertext 2 was a letter and not a number. The reasoning was that a sentence would most likely begin with a word not containing numbers.

In total, output for key length = 2,...,9 was generated and analyzed. Trying combinations for a key larger than 4 took too much time. The solution to this was to replace each corresponding k_i, where i = 4,5,6,7,8, with a simple dot ".". The idea was to generate a larger key containing some unknown parts in hope that an English sentence would be readable. Figure 5 shows partial outputs for key length 4 and 8. Common words found in **[1]** were manually searched for, but no indication of an English sentence could be found.

After a long search with no results, the previous assumption of a sentence not beginning with a number was dismissed. New outputs were generated for key length = 2,...,9 and analyzed manually. A new search for common words was performed but no results. A search of combinations of numbers



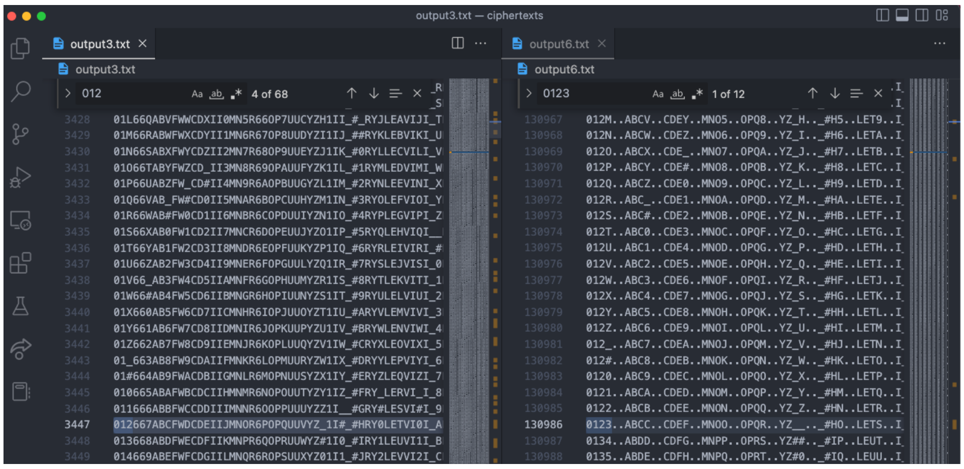
```
THE7NB80H5IS3TQJZNKCTHU2F30V9ZWO3T4ERFTSD4J3YL4F9I
THE8NB81H5IT3TQKZNKDTHU3F30W9ZWP3T4FRFTTD4J4YL4G9I
THE9NB82H5IU3TQLZNKETHU4F30X9ZWQ3T4GRFTUD4J5YL4H9I
THEANB83H5IV3TQMZNKFTHU5F30Y9ZWR3T4HRFTVD4J6YL4I9I
THEBNB84H5IW3TQNZNKGTHU6F30Z9ZWS3T4IRFTWD4J7YL4J9I
THECNB85H5IX3TQOZNKHTHU7F30_9ZWT3T4JRFTXD4J8YL4K9I
THEDNB86H5IY3TQPZNKITHU8F30#9ZWU3T4KRFTYD4J9YL4L9I
THEENB87H5IZ3TQQZNKJTHU9F3009ZWV3T4LRFTZD4JAYL4M9I
THEFNB88H5I_3TQRZNKKTHUAF3019ZWW3T4MRFT_D4JBYL4N9I
THEGNB89H5I#3TQSZNKLTHUBF3029ZWX3T4NRFT#D4JCYL4O9I
THEHNB8AH5I03TQTZNKMTHUCF3039ZWY3T4ORFT0D4JDYL4P9I
THEINB8BH5I13TQUZNKNTHUDF3049ZWZ3T4PRFT1D4JEYL4Q9I
THEJNB8CH5I23TQVZNKOTHUEF3059ZW_3T4QRFT2D4JFYL4R9I
THEKNB8DH5I33TQWZNKPTHUFF3069ZW#3T4RRFT3D4JGYL4S9I
THELNB8EH5I43TQXZNKQTHUGF3079ZW03T4SRFT4D4JHYL4T9I
THEMNB8FH5I53TQYZNKRTHUHF3089ZW13T4TRFT5D4JIYL4U9I
THENNB8GH5I63TQZZNKSTHUIF3099ZW23T4URFT6D4JJYL4V9I
THEONB8HH5I73TQ_ZNKTTHUJF30A9ZW33T4VRFT7D4JKYL4W9I
```

```
THE6....H5IR....ZNKB....F30U....3T4D....D4J2....9I
THE7....H5IS....ZNKC....F30V....3T4E....D4J3....9I
THE8....H5IT....ZNKD....F30W....3T4F....D4J4....9I
THE9....H5IU....ZNKE....F30X....3T4G....D4J5....9I
THEA....H5IV....ZNKF....F30Y....3T4H....D4J6....9I
THEB....H5IW....ZNKG....F30Z....3T4I....D4J7....9I
THEC....H5IX....ZNKH....F30_....3T4J....D4J8....9I
THED....H5IY....ZNKI....F30#....3T4K....D4J9....9I
THEE....H5IZ....ZNKJ....F300....3T4L....D4JA....9I
THEF....H5I_....ZNKK....F301....3T4M....D4JB....9I
THEG....H5I#....ZNKL....F302....3T4N....D4JC....9I
THEH....H5I0....ZNKM....F303....3T4O....D4JD....9I
THEI....H5I1....ZNKN....F304....3T4P....D4JE....9I
THEJ....H5I2....ZNKO....F305....3T4Q....D4JF....9I
THEK....H5I3....ZNKP....F306....3T4R....D4JG....9I
THEL....H5I4....ZNKQ....F307....3T4S....D4JH....9I
THEM....H5I5....ZNKR....F308....3T4T....D4JI....9I
THEN....H5I6....ZNKS....F309....3T4U....D4JJ....9I
```

Figure 5: Partial outputs of the program *bruteforceCipher2.py*

was done, starting with 0123 and 1234. An interest pattern appeared on outputs for key length 3 and 6, seen on figure 6. The first two halves of the lines showed patterns which seemed to align with a sequence of numbers, starting with 01234, and potentially continuing with the English alphabet. Moreover, output from key length 6 showed signs of English words, and the number of symbols up to where the English words began was counted to be 38, the same length of the given alphabet in the assignment. An assumption was therefore made that ciphertext 2 began with an encrypted version of the given alphabet.

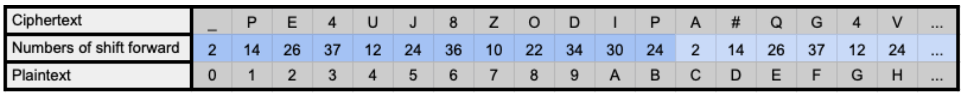
In order to verify this assumption, the beginning of ciphertext 2 was mapped to the given alphabet in order to show that a key of a certain key length would rotate and begin the cycle again. The result shown on figure 7 show that a key of length 12 was used, confirming that a Vigénère cipher was used. The key k = [2,14,26,37,12,24,36,10,22,34,30,24] was then used in a python program in order to decrypt the ciphertext. The results seen on figure 8 shows that ciphertext 2 was successfully decoded.



```
output3.txt — ciphertexts

output3.txt
012    4 of 68

01L66QABVFWWCDXII0MN5R66OP7UUCYZH1II_#_RYJLEAVIJI_T
01M66RABWFWXCDYII1MN6R67OP8UUDYZI1IJ_##RYKLEBVIKI_U
01N66SABXFWYCDZII2MN7R68OP9UUEYZJ1IK_#0RYLLECVILI_V
01O66TABYFWZCD_II3MN8R69OPAUUFYZK1IL_#1RYMLEDVIMI_W
01P66UABZFW_CD#II4MN9R6AOPBUUGYZL1IM_#2RYNLEEVINI_X
01Q66VAB_FW#CD0II5MNAR6BOPCUUHYZM1IN_#3RYOLEFVIOI_Y
01R66WAB#FW0CD1II6MNBR6COPDUUIYZN1IO_#4RYPLEGVIPI_Z
01S66XAB0FW1CD2II7MNCR6DOPEUUJYZO1IP_#5RYQLEHVIQI__
01T66YAB1FW2CD3II8MNDR6EOPFUUKYZP1IQ_#6RYRLEIVIRI_#
01U66ZAB2FW3CD4II9MNER6FOPGUULYZQ1IR_#7RYSLEJVISI_0
01V66_AB3FW4CD5IIAMNFR6GOPHUUMYZR1IS_#8RYTLEKVITI_1
01W66#AB4FW5CD6IIBMNGR6HOPIUUNYZS1IT_#9RYULELVIUI_2
01X660AB5FW6CD7IICMNHR6IOPJUUOYZT1IU_#ARYVLEMVIVI_3
01Y661AB6FW7CD8IIDMNIR6JOPKUUPYZU1IV_#BRYWLENVIWI_4
01Z662AB7FW8CD9IIEMNJR6KOPLUUQYZV1IW_#CRYXLEOVIXI_5
01_663AB8FW9CDAIIFMNKR6LOPMUURYZW1IX_#DRYYLEPVIYI_6
01#664AB9FWACDBIIGMNLR6MOPNUUSYZX1IY_#ERYZLEQVIZI_7
010665ABAFWBCDCIIHMNMR6NOPOUUTYZY1IZ_#FRY_LERVI_I_8
011666ABBFWCCDDIIIMNNR6OOPPUUUYZZ1I__#GRY#LESVI#I_9
012667ABCFWDCDEIIJMNOR6POPQUUVYZ_1I#_#HRY0LETVI0I_A
013668ABDFWECDFIIKMNPR6QOPRUUWYZ#1I0_#IRY1LEUVI1I_B
014669ABEFWFCDGIILMNQR6ROPSUUXYZ01I1_#JRY2LEVVI2I_C
```

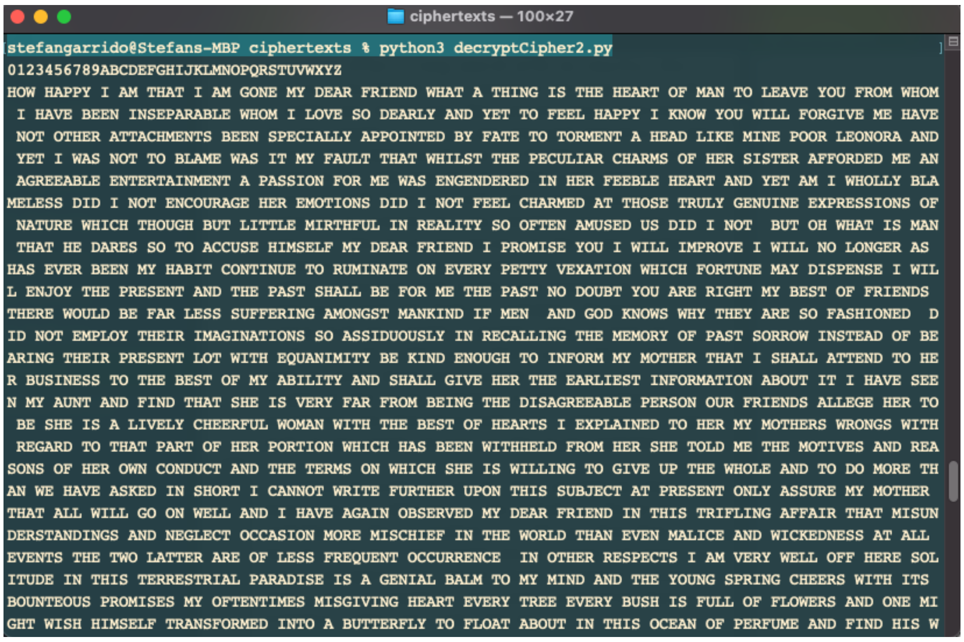
```
output6.txt

output6.txt
0123    1 of 12

012M..ABCV..CDEY..MNO5..OPQ8..YZ_H.._#H5..LET9..I_
012N..ABCW..CDEZ..MNO6..OPQ9..YZ_I.._#H6..LETA..I_
012O..ABCX..CDE_..MNO7..OPQA..YZ_J.._#H7..LETB..I_
012P..ABCY..CDE#..MNO8..OPQB..YZ_K.._#H8..LETC..I_
012Q..ABCZ..CDE0..MNO9..OPQC..YZ_L.._#H9..LETD..I_
012R..ABC_..CDE1..MNOA..OPQD..YZ_M.._#HA..LETE..I_
012S..ABC#..CDE2..MNOB..OPQE..YZ_N.._#HB..LETF..I_
012T..ABC0..CDE3..MNOC..OPQF..YZ_O.._#HC..LETG..I_
012U..ABC1..CDE4..MNOD..OPQG..YZ_P.._#HD..LETH..I_
012V..ABC2..CDE5..MNOE..OPQH..YZ_Q.._#HE..LETI..I_
012W..ABC3..CDE6..MNOF..OPQI..YZ_R.._#HF..LETJ..I_
012X..ABC4..CDE7..MNOG..OPQJ..YZ_S.._#HG..LETK..I_
012Y..ABC5..CDE8..MNOH..OPQK..YZ_T.._#HH..LETL..I_
012Z..ABC6..CDE9..MNOI..OPQL..YZ_U.._#HI..LETM..I_
012_..ABC7..CDEA..MNOJ..OPQM..YZ_V.._#HJ..LETN..I_
012#..ABC8..CDEB..MNOK..OPQN..YZ_W.._#HK..LETO..I_
0120..ABC9..CDEC..MNOL..OPQO..YZ_X.._#HL..LETP..I_
0121..ABCA..CDED..MNOM..OPQP..YZ_Y.._#HM..LETQ..I_
0122..ABCB..CDEE..MNON..OPQQ..YZ_Z.._#HN..LETR..I_
0123..ABCC..CDEF..MNOO..OPQR..YZ__.._#HO..LETS..I_
0134..ABDD..CDFG..MNPP..OPRS..YZ##.._#IP..LEUT..I_
0135..ABDE..CDFH..MNPQ..OPRT..YZ#0.._#IQ..LEUU..I_
```

Figure 6: Partial outputs of the program *bruteforceCipher2.py*



Ciphertext	_	P	E	4	U	J	8	Z	O	D	I	P	A	#	Q	G	4	V	...
Numbers of shift forward	2	14	26	37	12	24	36	10	22	34	30	24	2	14	26	37	12	24	...
Plaintext	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	...

Figure 7: Manual test showing the key used in ciphertext 2



```
ciphertexts — 100×27

stefangarrido@Stefans-MBP ciphertexts % python3 decryptCipher2.py
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
HOW HAPPY I AM THAT I AM GONE MY DEAR FRIEND WHAT A THING IS THE HEART OF MAN TO LEAVE YOU FROM WHOM
 I HAVE BEEN INSEPARABLE WHOM I LOVE SO DEARLY AND YET TO FEEL HAPPY I KNOW YOU WILL FORGIVE ME HAVE
 NOT OTHER ATTACHMENTS BEEN SPECIALLY APPOINTED BY FATE TO TORMENT A HEAD LIKE MINE POOR LEONORA AND
 YET I WAS NOT TO BLAME WAS IT MY FAULT THAT WHILST THE PECULIAR CHARMS OF HER SISTER AFFORDED ME AN
 AGREEABLE ENTERTAINMENT A PASSION FOR ME WAS ENGENDERED IN HER FEEBLE HEART AND YET AM I WHOLLY BLA
MELESS DID I NOT ENCOURAGE HER EMOTIONS DID I NOT FEEL CHARMED AT THOSE TRULY GENUINE EXPRESSIONS OF
 NATURE WHICH THOUGH BUT LITTLE MIRTHFUL IN REALITY SO OFTEN AMUSED US DID I NOT  BUT OH WHAT IS MAN
 THAT HE DARES SO TO ACCUSE HIMSELF MY DEAR FRIEND I PROMISE YOU I WILL IMPROVE I WILL NO LONGER AS
HAS EVER BEEN MY HABIT CONTINUE TO RUMINATE ON EVERY PETTY VEXATION WHICH FORTUNE MAY DISPENSE I WIL
L ENJOY THE PRESENT AND THE PAST SHALL BE FOR ME THE PAST NO DOUBT YOU ARE RIGHT MY BEST OF FRIENDS
THERE WOULD BE FAR LESS SUFFERING AMONGST MANKIND IF MEN  AND GOD KNOWS WHY THEY ARE SO FASHIONED  D
ID NOT EMPLOY THEIR IMAGINATIONS SO ASSIDUOUSLY IN RECALLING THE MEMORY OF PAST SORROW INSTEAD OF BE
ARING THEIR PRESENT LOT WITH EQUANIMITY BE KIND ENOUGH TO INFORM MY MOTHER THAT I SHALL ATTEND TO HE
R BUSINESS TO THE BEST OF MY ABILITY AND SHALL GIVE HER THE EARLIEST INFORMATION ABOUT IT I HAVE SEE
N MY AUNT AND FIND THAT SHE IS VERY FAR FROM BEING THE DISAGREEABLE PERSON OUR FRIENDS ALLEGE HER TO
 BE SHE IS A LIVELY CHEERFUL WOMAN WITH THE BEST OF HEARTS I EXPLAINED TO HER MY MOTHERS WRONGS WITH
 REGARD TO THAT PART OF HER PORTION WHICH HAS BEEN WITHHELD FROM HER SHE TOLD ME THE MOTIVES AND REA
SONS OF HER OWN CONDUCT AND THE TERMS ON WHICH SHE IS WILLING TO GIVE UP THE WHOLE AND TO DO MORE TH
AN WE HAVE ASKED IN SHORT I CANNOT WRITE FURTHER UPON THIS SUBJECT AT PRESENT ONLY ASSURE MY MOTHER
THAT ALL WILL GO ON WELL AND I HAVE AGAIN OBSERVED MY DEAR FRIEND IN THIS TRIFLING AFFAIR THAT MISUN
DERSTANDINGS AND NEGLECT OCCASION MORE MISCHIEF IN THE WORLD THAN EVEN MALICE AND WICKEDNESS AT ALL
EVENTS THE TWO LATTER ARE OF LESS FREQUENT OCCURRENCE  IN OTHER RESPECTS I AM VERY WELL OFF HERE SOL
ITUDE IN THIS TERRESTRIAL PARADISE IS A GENIAL BALM TO MY MIND AND THE YOUNG SPRING CHEERS WITH ITS
BOUNTEOUS PROMISES MY OFTENTIMES MISGIVING HEART EVERY TREE EVERY BUSH IS FULL OF FLOWERS AND ONE MI
GHT WISH HIMSELF TRANSFORMED INTO A BUTTERFLY TO FLOAT ABOUT IN THIS OCEAN OF PERFUME AND FIND HIS W
```

Figure 8: Partial output of the program *decryptCipher2.py*

Appendices

The following code corresponds to the program: mostCommonLetter.py

```python
import operator

try:
    ciphertext = open("cipher1.txt", "r")
except:
    print("Error reading file, check input!")
    exit()

#Remove newlines from file
ciphertext = ciphertext.read().replace('\n', '')
lst = {}

# Go through every char and count occurrences.
for char in ciphertext:
    if char in lst:
        lst[char] = lst[char] + 1
        continue
    lst[char] = 1

print()
print("Nr of unique symbols:", len(lst))
print(dict(sorted(lst.items(), key=operator.itemgetter(1),
    reverse=True)))
```

The following code corresponds to the program: mostCommonWord.py

```python
import operator

try:
    ciphertext = open("cipher1.txt", "r")
except:
    print("Error reading file, check input!")
    exit()

#Remove newlines from file
ciphertext = ciphertext.read().replace('\n', '')
#Split at character = #
cipher_splitted = ciphertext.split("#")
lst = {}

# Go through every word after splitting at # and count
    occurrences.
for word in cipher_splitted:
    if word == "":
        continue
    if word in lst:
        lst[word] = lst[word] + 1
        continue
    lst[word] = 1

print()
print("Nr of unique words:", len(lst))
print(dict(sorted(lst.items(), key=operator.itemgetter(1), reverse=True)))
```

The following code corresponds to the program: decryptCipher1.py

```python
def get_corresp_char(char):
    ascii_val = ord(char)

    #Special case for 0, # and _
    if ascii_val == 48:
        return "\n"
    if ascii_val == 35:
        return " "
    if ascii_val == 95:
        return "Z"

    # For all other chars move 1 step to the left
    ascii_val = ascii_val - 1
    return chr(ascii_val)

try:
    ciphertext = open("cipher1.txt", "r")
except:
    print("Error reading file, check input!")
    exit()

#Remove newlines from file
ciphertext = ciphertext.read().replace('\n', '')

#Go through every character: decrypt and print
for c in ciphertext:
    print(get_corresp_char(c), end = '')

print()
print("*** Decryption finished ***")
```

The following code corresponds to the program: bruteforceCipher2.py. Note: the code was modified for different key lengths.

```python
# define first line of ciphertext2 and corresponding used alphabet
orig_string = "_PE4UJ8Z0DIPA#QG4VK9_PU#MB0SG5WLA#4BYNTPKCJ03T4CGMMN05J072QC88AHC#GQ00"
symbols = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ_#"

# give every symbol in alphabet a corresponding value: 0,1,...,n
# and store results in char_to_val_list, val_to_char_list is the
# reversed key-val pair
char_to_val_list = {}
val_to_char_list = {}
counter = 0
for char in symbols:
    char_to_val_list[char] = counter
    val_to_char_list[counter] = char
    counter += 1

counter = 0
start = 0
current = 0
# The nested for loops go through each combination for key length = 4
# Assuming the actual key length is longer, each key lenght > 4 is
# handled by printing a ".", the output is then manually analysed
for k1 in range(start,38):
    for k2 in range(0,38):
        for k3 in range(0,38):
            for k4 in range(0,38):

                # The different keys alternate based on key length
                for char in orig_string:
                    # reset counter
                    if counter == 9:
                        counter = 0

                    # Go trough different printing options
                    if counter == 0:
                        current = k1
                    if counter == 1:
                        current = k2
                    if counter == 2:
                        current = k3
                    if counter == 3:
                        current = k4
                    if counter == 4 or counter == 5 or counter == 6 or counter == 7 or counter == 8:
                        print(".", end='')
                        counter +=1
                        continue

                    pos = (char_to_val_list[char]+current)%38
                    print(val_to_char_list[pos], end='')
                    counter += 1
                counter = 0
                print()
```

The following code corresponds to the program: decryptCipher2.py.

```python
symbols = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ_#"

try:
    ciphertext = open("cipher2.txt", "r")
except:
    print("Error reading file, check input!")
    exit()
```

```python

# give every symbol in alphabet a corresponding value: 0,1,...,n
# and store results in char_to_val_list, val_to_char_list is the
# reversed key-val pair
char_to_val_list = {}
val_to_char_list = {}
counter = 0
for char in symbols:
    char_to_val_list[char] = counter
    val_to_char_list[counter] = char
    counter += 1

#Remove newlines from file
ciphertext = ciphertext.read().replace('\n', '')

#key for decryption
key = [2,14,26,37,12,24,36,10,22,34,30,24]
current = 0

#Go through every character: decrypt and print
for char in ciphertext:

    if current == 12:
        current = 0

    pos = (char_to_val_list[char]+ key[current])%38

    if val_to_char_list[pos] == '_':
        print(' ', end='')
        current += 1
        continue

    if val_to_char_list[pos] == '#':
        print('\n', end='')
        current += 1
        continue

    print(val_to_char_list[pos], end='')
    current += 1
print()
print("*** Decryption finished ***")
```

References

[1] Wikipedia. Most common words in english. https://en.wikipedia.org/wiki/Most$_{c}ommon_{w}ords_{i}n_{E}nglish$.