2. a) Using frequency data from the webnotes (after correcting the frequency for b, which should be .015, not .01), we have these ciphertext character frequency distributions:

   Scheme A: letter 'e' should take 4 homophones. So:

   .081 .001 .027 .042 .032 .032 .032 .031 .022 .020 .061 .070 .002 .008 .040 .024 .067 .075 .019 .001 .060 .063 .091 .028 .010 .024 .002 .020 .001

   Scheme B: letter 'e' takes 3 homophones and letter 't' takes 2:

   .081 .001 .027 .042 .043 .042 .042 .022 .020 .061 .070 .002 .008 .040 .024 .067 .075 .019 .001 .060 .063 .046 .045 .028 .010 .024 .002 .020 .001

   Scheme C: letters 'a', 'c' and 't' each take 2 homophones:

   .041 .040 .001 .027 .042 .064 .063 .022 .020 .061 .070 .002 .008 .040 .024 .067 .075 .019 .001 .060 .063 .046 .045 .028 .010 .024 .002 .020 .001

   Which of these 3 distributions is smoothest? One measure of this is index of coincidence: the lower the index of coincidence, the smoother the distribution. You can use a python program from the course gitcode repo to compute the ioc of each of these: set C has the smallest ioc. So, as an encrypter, we would prefer Scheme C.

   b) In percent, before smoothing, we expect top 11 frequencies 13 9 8 7.5 7 6.5 6.5 6 6 4 4...

   So after A we expect the 13 is replaced by 4 counts of around 3.25, so the top 10 frequencies will be something like 9 8 7.5 7 6.5 6.5 6 6 4 4 ...

   after B, 13 is repaced by 4.3 4.3 4.3 and 9 by 4.5 4.5, so for top 10 we expect 8 7.5 7 6.5 6.5 6 6 4.5 4.5 4.3...

   after C, 13 is repaced by 6.5 6.5, 9 by 4.5 4.5, 8 by 4 4, so top 10 7.5 7 6.5 6.5 6.5 6.5 6 6 4 4...

   Here, the top 10 ctxt frequencies, in percent, are 9 9 9 6.7 6.7 5 5 5 5 4 ...

   It is hard to say which of these 3 fits the data best.

   c) we know that e,a,s have 2 homophones each, every other character has only 1. so maybe we will have luck by focussing on the most usually-frequent letter that has only 1 homophone: t. the good news is that t has an unusual digram frequency patter: th is very common. below is one way to crack this.

```
run freq/freq.py
from pairs, guess ctxt jx => th
abcdefghijklmnopqrstuvwxyz
.........t.............h..
jxyimzljxtiqctouzhjxzjjxeczifxtbyffulmzlrhtqaydwjxtsyfxthevcqhogkutdevlsejl
th.....th.........th.tth.....h..................th....h..................t.



guess 'that', so ctxt z => a
abcdefghijklmnopqrstuvwxyz
.........t.............h.a
jxyimzljxtiqctouzhjxzjjxeczifxtbyffulmzlrhtqaydwjxtsyfxthevcqhogkutdevlsejl
th...a.th.......a.thatth..a..h........a.........th....h..................t.
```

```
(guessing ctxt x is h) so frequent ctxt xt => he, so t => e
abcdefghijklmnopqrstuvwxyz
.........t.........e...h.a
jxyimzljxtiqctouzhjxzjjxeczifxtbyffulmzlrhtqaydwjxtsyfxthevcqhogkutdevlsejl
th...a.the...e..a.thatth..a..he......a...e.....the...he.........e......t.

ctxt y probably vowel, try y => i
abcdefghijklmnopqrstuvwxyz
.........t.........e...hia
jxyimzljxtiqctouzhjxzjjxeczifxtbyffulmzlrhtqaydwjxtsyfxthevcqhogkutdevlsejl
thi..a.the...e..a.thatth..a..he.i.....a...e..i..the.i.he.........e......t.

guess 1st word 'this' so i => s
abcdefghijklmnopqrstuvwxyz
........st.........e...hia
jxyimzljxtiqctouzhjxzjjxeczifxtbyffulmzlrhtqaydwjxtsyfxthevcqhogkutdevlsejl
this.a.thes..e..a.thatth..as.he.i.....a...e..i..the.i.he.........e......t.

guess 2nd word 'was' so mzl => was
abcdefghijklmnopqrstuvwxyz
........st.sw......e...hia
jxyimzljxtiqctouzhjxzjjxeczifxtbyffulmzlrhtqaydwjxtsyfxthevcqhogkutdevlsejl
thiswasthes..e..a.thatth..as.he.i...swas..e..i..the.i.he.........e...s..ts

have guessed txztyiml, maybe next most frequent ctxt character e => o
abcdefghijklmnopqrstuvwxyz
....o...st.sw......e...hia
jxyimzljxtiqctouzhjxzjjxeczifxtbyffulmzlrhtqaydwjxtsyfxthevcqhogkutdevlsejl
thiswasthes..e..a.thattho.as.he.i...swas..e..i..the.i.he.o........e.o.s.ots

guess 'thomas' and 'scots' so cs => mc
abcdefghijklmnopqrstuvwxyz
..m.o...st.sw.....ce...hia
jxyimzljxtiqctouzhjxzjjxeczifxtbyffulmzlrhtqaydwjxtsyfxthevcqhogkutdevlsejl
thiswasthes.me..a.thatthomas.he.i...swas..e..i..theci.he.o.m......e.o.scots

guess 'cipher'  so fh => ph
abcdefghijklmnopqrstuvwxyz
..m.op.rst.sw.....ce...hia
jxyimzljxtiqctouzhjxzjjxeczifxtbyffulmzlrhtqaydwjxtsyfxthevcqhogkutdevlsejl
thiswasthes.me..arthatthomasphe.ipp.swas.re..i..theciphero.m.r....e.o.scots

guess 'phelippes ofmaryqueenofscots' buvqogkd => lefayqun
abcdefghijklmnopqrstuvwxyz
.lmnopqrstusw.y.a.ceef.hia
jxyimzljxtiqctouzhjxzjjxeczifxtbyffulmzlrhtqaydwjxtsyfxthevcqhogkutdevlsejl
thiswasthesameyearthatthomasphelippeswas.rea.in.thecipherofmaryqueenofscots
```

```
finally 'breaking'
abcdefghijklmnopqrstuvwxyz
klmnopqrstusw.y.abceefghia
jxyimzljxtiqctouzhjxzjjxeczifxtbyffulmzlrhtqaydwjxtsyfxthevcqhogkutdevlsejl
thiswasthesameyearthatthomasphelippeswasbreakingthecipherofmaryqueenofscots
```

3. i) (5*4 + 2*1 + 3*2)/(10*9) = (20 + 2 + 6)/90 = 28/90 = 0.3111...

   ii) (3*2 + 2*1 + 2*1 + 2*1)/(9*8) = (6 + 2 + 2 + 2)/72 = 12/72 = 0.1666...

   iii) (5*3 + 2*2 + 3*2 + 0*2)/(10*9) = (15 + 4 + 6 + 0)/90 = 25/90 = 0.2777...

   iv) (5/10)*0.081 + (2/10)*0.001 + (3/10)*0.027 = 0.0405 + 0.0002 + 0.0081 = 0.0488... Every other character contributes 0 to the imc and so can be omitted.

4. i) npojgwym hbhomw betcsx

   ii) the additions that need to be performed with keyword `babyface` are 1 0 1 -1 5 0 2 4 respectively. these are easy to do by examining an alphabet and counting. the additions for keyword`viginere` are -5 8 7 8 13 4 -9 4. performing these shifts by hand is more likely to result in errors, since the shifts are greater. so prefer `babyface`. (and hey! why did no one comment on the misspelling of vigenere?)

5. i) Here is the output from running `kgrams.py`. Beside each number I have listed factors. The most common keyword length is 3. this is supported by the Babbage/Kasiski test, but **since there are no repeated kgrams for k at least 3, we would expect many false positives, and so we would expect this test might be unreliable for this case**.

```
check for repeated 2 -grams
103 za          103
21 dp           3 7 21
93 pg           3 31
87 ae           3 19
23 ei           23
48 il           2 3 4 6 8 12 24 48
82 ey           2 41
24 pp           2 3 4 6 8 12 24
66 pc           2 3 33
48 op           2 3 4 6 8 12 24 48
43 hp           43
53 sh           53
24 ti           2 3 4 6 8 12 24
16 gp           2 4 8 16
check for repeated 3 -grams
```

   ii) the program suggests the most probable length is 8, with most likely key `polyaleh`. this is supported by the ioc data: length 8 has ioc .064, the next best shift has ioc .05. since the correct shift would give an expected ioc close to that of English, about .065, there is only 1 obvious candidate here: keyword length 8.

   iii) the suggested shift of the 2nd last character is shift 4 (keyletter e), with English-imc .075. **Notice that there is one other strong candidate:** shift 15 (keyletter p) with English-imc .070.

   Shift 4 gives keyword `polyaleh`. Shift 15 gives keyword `polyalph`.

   iv) The best key found by the program is **not** the correct key.

   Here is text deciphered using key `polyaleh`:

```
babbagpw asinsptr edtoatee mptadeni phermeyt byanexnh angeofwe tterswtt
hthwaiee sadentts tfrombci stolwieh aratheci nnocenev iewofctp hers
```

This almost makes sense: 7 out of 8 characters make sense.

`babbagpw asinsptr edtoatee mptadeni` should probably be

`babbagew asinspir edtoatte mptadeci`.

The key suggested in part iii) — `polyalph`, a prefix of polyalphabetic, is the correct key. The plaintext is

`babbagewasinspiredtoattemptadeciphermentbyanexchangeofletters`

`withthwaitesadentistfrombristolwitharatherinnocentviewofciphers`

Both choices in iii) had very high imc with English, one by coincidence, and one because it was correct.

iv)

`lp` + `ec` yields `ei` (1st occurence)

`al` + `ex` yields `ei` (2nd occurence)

So the false positive is as shown below

```
polyalphpolyalphpolyalphpolyalphpolyalphpolyal  keyword
babbagewasinspiredtoattemptadeciphermentbyanex  plaintext
qomzartdpgtlsaxytremaeilbdeydprpevppmpcaqmllei  ciphertext
```

```
.....lp.  ....al..   keyword
.....tt.  ....ex..   plaintext
.....ei.  ....ei..   ciphertext
```