Caverphone Revisited

Caversham Project Occasional Technical Paper **Code Number**:CTP150804
Author: David Hood **Original Publication Date:**15th August 2004
Keywords: phonetic encoding, procedures, comparisons **Last Updated:** 10th December 2004
E-mail: caversham@otago.ac.nz
Original location: http://caversham.otago.ac.nz/files/working/ctp150804.pdf

The Technical papers series is intended to document specific technical solutions to problems encountered within the project. It is being made available to the public in the event the techniques outlined may be of use to others.

Synopsis: The original Caverphone algorithm was provide to document the phonetic matching procedures used in linking the Caversham Project databases, and to give some guidelines for those attempting similar projects. It was not intended as a general phonetic matching code. Version two of Caverphone has been created as a general purpose English phonetic matching system.

Description of the problem:
When the original caversham phonetic matching algorithm was made available[1], it was intended to document the process used by the project and to provide some guidelines for others undertaking similar tasks. Because the rules phonetic matching were intended for a very specific database, the process of matching was able to draw on characteristics of the specific data to supplement a 'good enough' set of phonetic matching rules, rather than treating it as an exercise in pure phonetic matching. However, since publication of the original Caverphone working paper there has been an interest in this approach of a cascading series of rules to other data sets. The original rules were, in my opinion, not suited to this. Caverphone 2.0 has been developed as a more general phonetic match.

Procedure:
In creating a general purpose English language matching version of Caverphone from the original, a number of changes were made to it. However, as with the first version, a theoretical implementation would begin with a list of potential words that the word in question could be matched to. This word list would have both the true spelling and the phonetic code of each word stored. When looking for matches to a target word, the target word is converted into its phonetic code. The word list is then searched, and all entries with the same phonetic code are produced as a result.

Caverphone 2.0 is being made available for free use for the benefit of anyone who has a use for it, with the proviso that the Caversham Project at the University of Otago should be acknowledge as the original source.

The steps in Caverphone 2.0 are

Start with a word
Convert to lowercase
Remove anything not in the standard alphabet (typically a-z)[2]
Remove final e
If the name starts with cough make it cou2f

1. Available from http://caversham.otago.ac.nz/files/working/ctp060902.pdf
2. This may vary if the set of letters includes characters such as æ, ā, or ø

If the name starts with rough make it rou2f
If the name starts with tough make it tou2f
If the name starts with enough make it enou2f
If the name starts with enough make it trou2f
If the name starts with gn make it 2n
If the name ends with mb make it m2
replace cq with 2q
replace ci with si
replace ce with se
replace cy with sy
replace tch with 2ch
replace c with k
replace q with k
replace x with k
replace v with f
replace dg with 2g
replace tio with sio
replace tia with sia
replace d with t
replace ph with fh
replace b with p
replace sh with s2
replace z with s
replace an initial vowel[3] with an A
replace all other vowels with a 3
replace j with y
replace an initial y3 with Y3
replace an initial y with A
replace y with 3
replace 3gh3 with 3kh3
replace gh with 22
replace g with k
replace groups of the letter s with a S
replace groups of the letter t with a T
replace groups of the letter p with a P
replace groups of the letter k with a K
replace groups of the letter f with a F
replace groups of the letter m with a M
replace groups of the letter n with a N
replace w3 with W3
replace wh3 with Wh3
if the name ends in w replace the final w with 3
replace w with 2
replace an initial h with an A
replace all other occurrences of h with a 2
replace r3 with R3
if the name ends in r replace the replace final r with 3

---

3. Vowels are normally a,e,i,o,u but depending on the data might include characters such as æ, ā, or ø

replace r with 2
replace l3 with L3
if the name ends in l replace the replace final l with 3
replace l with 2
remove all 2s
if the name end in 3, replace the final 3 with A
remove all 3s
put ten 1s on the end
take the first ten characters as the code

The key aspects of the algorithm are that vocalic and non-vocalic versions of consonants are blended together, vowels are significant if initial or final (though they will also affect the coding of surrounding letters).

I decided that a fixed length code would be useful for subsequent work on transcription errors, and ten places was decided on as being a length sufficient for every name that I could test it on.

Examples:

Stevenson - take a name
stevenson - convert to lower case
stevenson - remove nonstandard letters
stefenson - replace v with f
st3f3ns3n - replace vowels with 3
St3f3nS3n - replace s with S
ST3f3nS3n - replace t with T
ST3F3nS3n - replace f with F
ST3F3NS3N- replace n with N
STFNSN - remove all 3s
STFNSN1111- add ones

Peter - take a name
peter - convert to lower case
peter - remove nonstandard letters
p3t3r - replace vowels with 3
p3T3r - replace t with T
P3T3r - replace p with P
P3T33 - replace final r with 3
P3T3A - replace final 3 with A
PTA - remove all 3s
PTA1111111- add ones

Testing:

To get a sense of the number of potential matches verses number of false positive results when matching, this algorithm was compared to soundex and metaphone[4] using files from the Moby word list collection[5]. I have included my own python implementation of the matching

4. This paper used the implementations of soundex and metaphone present in php 4.x
5. Project Gutenberg etext 3201, http://www.gutenberg.net/etext/3201

algorithm as a appendix. Note that this is written for ease of reading the algorithm, not for efficient implementation.

Tests were run using the Moby list of 1000 most frequent words and the Moby surnames list. Prior to running the tests the files were edited to remove apparent duplicates, this reduced the frequent words list to 900 entries.

Moby list of 1000 frequently used words:

Number of unique codes:
Soundex: 552 codes, Metaphone 680 codes, Caverphone 542 codes

Among this small sample Metaphone has the most unique codes, while Caverphone has marginally less than Soundex.

Distribution of codes:

Table 1: Percentage distribution of number of entries per code for Moby frequent words list

Entries per Code	Soundex distribution	Caverphone distribution	Metaphone distribution
1	63.41	76.38	81.91
2	22.64	10.52	10.59
3	7.07	4.80	3.68
4	3.80	2.40	2.06
5	1.27	1.85	1.32
6	1.09	1.11	–
7	0.72	0.55	0.15
8	–	0.55	0.29
9	–	0.37	–
10	–	0.37	–
11	–	0.55	–
12	–	0.18	–
13	–	0.18	–
14	–	–	–
15	–	0.18	–
total codes	552	542	680

Soundex's reputation for being a loose match compared to Metaphone seems to stem from the number of unique entries (the early part of the distribution) rather than the tail of the distribution (which is similar in both cases). While the tail of the Caverphone distribution extends to more shared codes than either other distribution, the distribution itself is close to that of Metaphone.

Commonest codes were:
Soundex: (7 entries) L200, R300, T200, T600
L200 words were: leach, leg, less, like, look
Caverphone: (15 entries) AT11111111
AT11111111 words were: add, aid, at, art, eat, earth, head, hit, hot, hold, hard, heart, it, out, old
Metaphone: (8 entries) FR, RT

FR words were: far, fear, fire, for, four, free, vary, very

Matches of randomly selected words:

Word: ready, Soundex: R300, Caverphone: RTA1111111, Metaphone: RT
Soundex matches: radio, rate, read, ready, red, ride, road
Caverphone matches: rather, ready, writer
Metaphone matches: radio, rate, read, ready, red, ride, road, write
Commentary: Caverphone's sensitivity to the final syllable produced significantly different matches.

Word: social, Soundex: S240, Caverphone: SSA1111111, Metaphone: SXL
Soundex matches: social
Caverphone matches: social
Metaphone matches: social
Commentary: This example happens to be one where all three codes do not share words.

Word: able, Soundex: A140, Caverphone: APA1111111, Metaphone: ABL
Soundex matches: able, apply
Caverphone matches: able, appear
Metaphone matches: able
Commentary: Here is an example of Metaphone returning fewer matches than the other codes.

Moby List of Surnames (21986 common surnames):

Number of unique codes:
Soundex: 2911 codes, Caverphone 4339 codes, Metaphone 6791 codes

Distribution of codes:
Table 2: Percentage distribution of number of entries per code for Moby surname list

Entries per Code	Soundex distribution	Caverphone distribution	Metaphone distribution
1	25.21	48.24	56.31
2	14.19	15.49	15.92
3	9.69	8.60	7.48
4	7.59	5.23	4.67
5	6.53	3.41	3.40
6	4.33	2.19	2.06
7	3.64	2.42	1.72
8	3.81	1.45	1.00
9	3.13	1.20	1.10
10	2.68	1.29	0.80
11-15	7.21	3.50	2.33
16-20	3.85	1.87	1.22
21-30	3.54	2.19	0.96
31-50	3.09	1.59	0.63
51+	1.51	1.34	0.40
total codes	2911	4339	6791

Using a large set of material that the three coding systems were designed for, the distribution of Caverphone still lies between Soundex, and Metaphone (though closer to Metaphone than when using the smaller set of data).

Though not obvious from the table, while the distribution of the tail is shallower for Caverphone than Soundex, it is also longer as can be seen in the maximum number of names per code with each system.

Commonest codes were:
Soundex: D500 113 entries
Metaphone: TN 115 entries
Caverphone: ATA1111111 174 codes

Matches of randomly selected names:
Tedder Soundex:T360 Metaphone:TTR Caverphone:TTA1111111
Soundex: Teador, Tedder, Tedra, Teeter, Teodoor, Teodor, Teodora, Teodoro, Theadora, Theodor, Theodora, Theodore, Tuddor, Tudor
Metaphone: Daughtry, Dedra, Deidre, Didier, Dieter, Ditter, Dituri, Teador, Tedder, Tedra, Teeter, Teodoor, Teodoor, Teodor, Teodora, Teodoro, Tuddor, Tudor
Caverphone: Darda, Datha, Dedie, Deedee, Deerdre, Deidre, Deirdre, Detta, Didi, Didier, Dido, Dierdre, Dieter, Dita, Ditter, Dodi, Dodie, Dody, Doherty, Dorthea, Dorthy, Doti, Dotti, Dottie, Dotty, Doty, Doughty, Douty, Dowdell, Duthie, Tada, Taddeo, Tadeo, Tadio, Tati, Teador, Tedda, Tedder, Teddi, Teddie, Teddy, Tedi, Tedie, Teeter, Teodoor, Teodor, Terti, Theda, Theodor, Theodore, Theta, Thilda, Thordia, Tilda, Tildi, Tildie, Tildy, Tita, Tito, Tjader, Toddie, Toddy, Torto, Tuddor, Tudor, Turtle, Tuttle, Tutto

Karleen Soundex:K645 Metaphone:KRLN Caverphone:KLN1111111
Soundex: Kara-Lynn, Karalynn, Karilynn, Karlan, Karleen, Karlen, Karlene, Karlens, Karlin, Karlyn, Karolina, Karoline, Karolyn, Karylin, Koerlin, Krahling, Kurland
Metaphone: Carilyn, Carleen, Carlen, Carlene, Carlin, Carlina, Carline, Carlyn, Carlynn, Carlynne, Carolan, Carolann, Carolin, Carolina, Caroline, Carolyn, Carolyne, Carolynn, Carolynne, Coraline, Coralyn, Crelin, Crellen, Garlan, Garlen, Gorlin, Kara-Lynn, Karalynn, Karilynn, Karlan, Karleen, Karlen, Karlene, Karlin, Karlyn, Karolina, Karoline, Karolyn, Karylin, Koerlin
Caverphone: Cailean, Calan, Calen, Callahan, Callan, Callean, Carleen, Carlen, Carlene, Carlin, Carline, Carlyn, Carlynn, Carlynne, Charlean, Charleen, Charlene, Charline, Cherlyn, Chirlin, Clein, Cleon, Cline, Cohleen, Colan, Coleen, Colene, Colin, Colleen, Collen, Collin, Colline, Colon, Cullan, Cullen, Cullin, Gaelan, Galan, Galen, Garlan, Garlen, Gaulin, Gayleen, Gaylene, Giliane, Gillan, Gillian, Glen, Glenn, Glyn, Glynn, Gollin, Gorlin, Kalin, Karlan, Karleen, Karlen, Karlene, Karlin, Karlyn, Kaylyn, Keelin, Kellen, Kellene, Kellyann, Kellyn, Khalin, Kilan, Kilian, Killen, Killian, Killion, Klein, Kleon, Kline, Koerlin, Kylen, Kylynn, Quillan, Quillon, Qulllon, Xylon

Dyun Soundex:D500 Metaphone:TYN Caverphone:TN11111111
Soundex: Dam, Dame, Dan, Dana, Danae, Dane, Daney, Dani, Dania, Danie, Danieu, Dann, Danna, Danni, Dannie, Danny, Dannye, Danya, Daune, Dawn, Dawna, Dayna, Ddene, Dean, Deana, Deane, Deanna, Deanne, DeeAnn, Deeann, Deeanne, Deena, Deenya, Deeyn, Deina, Demmy, Demy, Den, Dena, Denae, Dene, Deni, Denie, Denn, Denna, Denney, Denni, Dennie, Denny, Deny, Deonne, Deuno, Dewain, Dewayne, Dhumma, Diahann, Dian, Diana, Diane, Diann, Dianna, Dianne, Diannne, Diena, Dina, Dinah, Dine, Dinnie, Dinny, Dino, Dion, Dione, Dionne, Doane, Doehne, Dom, Don, Dona, Donahoe, Donahue, Donia, Donn, Donna, Donni, Donnie, Donny, Donoho, Donohue, Doone, Down, Downe, Downey, Duane, Duma, Dumah, Dumm, Dun, Dunn, Duyne, Dwain, Dwaine, Dwan, Dwane, Dwayne, Dyan, Dyana, Dyane, Dyann, Dyanna, Dyanne, Dyna, Dynah, Dyun
Metaphone: Dyan, Dyana, Dyane, Dyann, Dyanna, Dyanne, Dyun
Caverphone: Dan, Dane, Dann, Darn, Daune, Dawn, Ddene, Dean, Deane, Deanne, DeeAnn, Deeann, Deeanne, Deeyn, Den, Dene, Denn, Deonne, Diahann, Dian, Diane, Diann, Dianne, Diannne, Dine,

Dion, Dione, Dionne, Doane, Doehne, Don, Donn, Doone, Dorn, Down, Downe, Duane, Dun, Dunn, Duyne, Dyan, Dyane, Dyann, Dyanne, Dyun, Tan, Tann, Teahan, Ten, Tenn, Terhune, Thain, Thaine, Thane, Thanh, Thayne, Theone, Thin, Thorn, Thorne, Thun, Thynne, Tien, Tine, Tjon, Town, Towne, Turne, Tyne

To assess the strengths of all three algorithms at making true positive matches I compared the codes generated by 393 known name variations within the records of the Caversham project (for the test file see http://caversham.otago.ac.nz/files/variantNames.csv). This is in no sense a 'blind' test as the name variation was established using a combination of hand-matching and linking using the first version of the Caverphone algorithm. As a result of this Caverphone 2.0 was expected to perform well at matching different versions of the same name

Of the 393 name pairs Soundex matched 328 of them (83.50%), Metaphone matched 303 pairs (77.10%), and Caverphone matched 373 pairs (94.91%). From these results it is indicated that Metaphone's reduced false positives when compared to Soundex does come at a cost of decreased successful matches.

In glancing through those matches solely made by Caverphone, the successes seem to come in cases of differing initial vowels and in looser handling of post-vocalic consonants than metaphone. Those matches missed by all three seem to be the hand corrections where the variation is the result of transcription rather than phonetic error.

While Caverphone's sensitivity to false positives lies between Metaphone and Soundex (tending to be similar to Metaphone), it does seem to be better at including true positives within the set of results. In using this within the project, phonetic codes are used when we cannot find a match by searching with other criteria. Comparing unmatched entries in two sources reduces the number of false positives dramatically than matching against a complete source. This process of potential match minimization prior to phonetic testing enables much greater automation of the process.

Appendix 1: Python Caverphone implementation used in testing (non-optimised).

```
def caverphone(wordToConvert):
      # produce a Caverphone code of a word

      # make the word lowercase
      workingCopyOfWordToConvert = string.lower(wordToConvert)

      # remove nonstandard characters
      partsOfName = re.split('[^a-z]+',workingCopyOfWordToConvert)
      workingCopyOfWordToConvert = string.join(partsOfName,'')

      # remove final e
      if workingCopyOfWordToConvert.endswith('e'):
            workingCopyOfWordToConvert = workingCopyOfWordToConvert[0:-1]

      # If the name starts with cough make it cou2f
      if workingCopyOfWordToConvert.startswith('cough'):
            workingCopyOfWordToConvert = 'cou2f' + workingCopyOfWordToConvert[5:]

      # If the name starts with rough make it rou2f
      if workingCopyOfWordToConvert.startswith('rough'):
            workingCopyOfWordToConvert = 'rou2f' + workingCopyOfWordToConvert[5:]
```

```
# If the name starts with tough make it tou2f
if workingCopyOfWordToConvert.startswith('tough'):
     workingCopyOfWordToConvert = 'tou2f' + workingCopyOfWordToConvert[5:]

# If the name starts with enough make it enou2f
if workingCopyOfWordToConvert.startswith('enough'):
     workingCopyOfWordToConvert = 'enou2f' + workingCopyOfWordToConvert[6:]

# If the name starts with trough make it trou2f
if workingCopyOfWordToConvert.startswith('trough'):
     workingCopyOfWordToConvert = 'trou2f' + workingCopyOfWordToConvert[6:]

# If the name starts with gn make it 2n
if workingCopyOfWordToConvert.startswith('gn'):
     workingCopyOfWordToConvert = '2n' + workingCopyOfWordToConvert[2:]

# If the name ends with mb make it m2
if workingCopyOfWordToConvert.endswith('mb'):
     workingCopyOfWordToConvert = workingCopyOfWordToConvert[0:-1] + '2'

# replace cq with 2q
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('cq','2q')

# replace ci with si
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('ci','si')

# replace ce with se
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('ce','se')

# replace cy with sy
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('cy','sy')

# replace tch with 2ch
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('tch','2ch')

# replace c with k
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('c','k')

# replace q with k
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('q','k')

# replace x with k
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('x','k')

# replace v with f
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('v','f')
```

```
# replace dg with 2g
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('dg','2g')

# replace tio with sio
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('tio','sio')

# replace tia with sia
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('tia','sia')

# replace d with t
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('d','t')

# replace ph with fh
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('ph','fh')

# replace b with p
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('b','p')

# replace sh with s2
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('sh','s2')

# replace z with s
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('z','s')

# replace an initial vowel with an A
if workingCopyOfWordToConvert.startswith('a'):
     workingCopyOfWordToConvert = 'A' + workingCopyOfWordToConvert[1:]
if workingCopyOfWordToConvert.startswith('e'):
     workingCopyOfWordToConvert = 'A' + workingCopyOfWordToConvert[1:]
if workingCopyOfWordToConvert.startswith('i'):
     workingCopyOfWordToConvert = 'A' + workingCopyOfWordToConvert[1:]
if workingCopyOfWordToConvert.startswith('o'):
     workingCopyOfWordToConvert = 'A' + workingCopyOfWordToConvert[1:]
if workingCopyOfWordToConvert.startswith('u'):
     workingCopyOfWordToConvert = 'A' + workingCopyOfWordToConvert[1:]

# replace all other vowels with a 3
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('a','3')
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('e','3')
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('i','3')
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('o','3')
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('u','3')

# replace j with y
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('j','y')
```

```
# replace an initial y3 with Y3
if workingCopyOfWordToConvert.startswith('y3'):
     workingCopyOfWordToConvert = 'Y3' + workingCopyOfWordToConvert[2:]

# replace an initial y with A
if workingCopyOfWordToConvert.startswith('y'):
     workingCopyOfWordToConvert = 'A' + workingCopyOfWordToConvert[1:]

# replace y with 3
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('y','3')

# replace 3gh3 with 3kh3
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('3gh3','3kh3')

# replace gh with 22
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('gh','22')

# replace g with k
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('g','k')

# replace groups of the letter s with a S
otherPartsOfName = re.split('s+',workingCopyOfWordToConvert)
workingCopyOfWordToConvert = string.join(otherPartsOfName,'S')

# replace groups of the letter t with a T
otherPartsOfName = re.split('t+',workingCopyOfWordToConvert)
workingCopyOfWordToConvert = string.join(otherPartsOfName,'T')

# replace groups of the letter p with a P
otherPartsOfName = re.split('p+',workingCopyOfWordToConvert)
workingCopyOfWordToConvert = string.join(otherPartsOfName,'P')

# replace groups of the letter k with a K
otherPartsOfName = re.split('k+',workingCopyOfWordToConvert)
workingCopyOfWordToConvert = string.join(otherPartsOfName,'K')

# replace groups of the letter f with a F
otherPartsOfName = re.split('f+',workingCopyOfWordToConvert)
workingCopyOfWordToConvert = string.join(otherPartsOfName,'F')

# replace groups of the letter m with a M
otherPartsOfName = re.split('m+',workingCopyOfWordToConvert)
workingCopyOfWordToConvert = string.join(otherPartsOfName,'M')

# replace groups of the letter n with a N
otherPartsOfName = re.split('n+',workingCopyOfWordToConvert)
workingCopyOfWordToConvert = string.join(otherPartsOfName,'N')
```

```
# replace w3 with W3
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('w3','W3')

# replace wh3 with Wh3
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('wh3','Wh3')

# if the name ends in w replace the final w with 3
if workingCopyOfWordToConvert.endswith('w'):
     workingCopyOfWordToConvert = workingCopyOfWordToConvert[0:-1] + '3'

# replace w with 2
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('w','2')

# replace an initial h with an A
if workingCopyOfWordToConvert.startswith('h'):
     workingCopyOfWordToConvert = 'A' + workingCopyOfWordToConvert[1:]

# replace all other occurrences of h with a 2
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('h','2')

# replace r3 with R3
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('r3','R3')

# if the name ends in r replace the replace final r with 3
if workingCopyOfWordToConvert.endswith('r'):
     workingCopyOfWordToConvert = workingCopyOfWordToConvert[0:-1] + '3'

# replace r with 2
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('r','2')

# replace l3 with L3
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('l3','L3')

# if the name ends in l replace the replace final l with 3
if workingCopyOfWordToConvert.endswith('l'):
     workingCopyOfWordToConvert = workingCopyOfWordToConvert[0:-1] + '3'

# replace l with 2
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('l','2')

# remove all 2s
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('2','')

# if the name end in 3, replace the final 3 with A
if workingCopyOfWordToConvert.endswith('3'):
     workingCopyOfWordToConvert = workingCopyOfWordToConvert[0:-1] + 'A'
```

```
# remove all 3s
workingCopyOfWordToConvert = workingCopyOfWordToConvert.replace('3','')

# put ten 1s on the end
workingCopyOfWordToConvert = workingCopyOfWordToConvert + '1111111111'

# take the first ten characters as the code
workingCopyOfWordToConvert = workingCopyOfWordToConvert[0:10]

return workingCopyOfWordToConvert
```