Language processing guidelines

General guidelines

- Every language has an "owner": a single person responsible of handling that language. This person is in charge of updating the documentation as necessary, and will contact others to consult and perform individual parts.
- Our interest in languages is dictated by (a) the availability of word lists (e.g. from Crubadan, Open Subtitles), (b) the transparency of the alphabet, (c) language family diversity, and (d) good documentation on the phonology of the language.
- All the files should be saved in utf8 encoding

Language processing stages

Data collection

- Collect all major available data from public sources. Internet source should be printed to PDF. Prominent sources include
  - Ethnologue, e.g. https://www.ethnologue.com/language/quz
  - Glottologue, e.g. https://glottolog.org/resource/languoid/id/cusc1236
  - Wikipedia, e.g. https://en.wikipedia.org/wiki/Cusco_Quechua

  Ideally, those sources should not be used directly, but refer to other sources that can more legitimately be cited (books, articles).

  A decent source for verification is Wiktionary, which often contains a number of words and their pronunication in different languages. Such form could be fairly annotation specific (at least in English there are several options) and are likely to be phonetic rather than phonemic
- All pdfs, scanned and otherwise, should be kept under /Scans/pdfs/*Language code and name*, e.g. `Scans/pdfs/ro_Romanian`
- Every resource should be documented in .bib format in Scans/pdfs/*Language code and name*.pdf. Notice that urls have a special format that includes urldate (and are otherwise `@misc` resources). Zotero can be used to handle most of this work, especially for urls. A .bib file is a text file that contains bibliographic entries such as:

```
@misc{GlottologCuscoQuechua,
    title = {Glottolog 3.3 - {Cusco} {Quechua}},
    url = {https://glottolog.org/resource/languoid/id/cusc1236},
    urldate = {2018-09-17},
}

@Book{Zipf1935,
Title                    = {The Psycho-biology of Language: an Introduction to Dynamic Philology},
Author                   = {Zipf, George Kingsley},
Publisher                = {Houghton, Mifflin},
Year                     = {1935}
}

@Article{Zipf1929,
Title                    = {{R}elative frequency as a determinant of phonetic change},
Author                   = {Zipf, George Kingsley},
Journal                  = {{H}arvard {S}tudies in {C}lassical {P}hilology},
Year                     = {1929},
Pages                    = {1--95},
Volume                   = {15}
}
```

- Documentation should be written in Rmarkdown (or plain markdown). If you are not comfortable with markdown (yet), use plain html, in a way that would be easily translatable to markdown (i.e. nothing fancy). A sample document may look like this:

  ```
  ---
  title: Cusco Quechua
  ---

  # Background

  **Language family**: Quechuan / Quechua II / Southern Quechua / Cusco Quechua

  ...

  # Phonology

  ## Consonants

  ...

  ## Vowels

  ...

  # Alphabet

  ...

  # Rules

  (an abstract representation of the rules you come up with)
  ```

- A complete language should have at least the following files

  ***langcode*.Rmd** The documentation of the language

  ***langcode*.bib** The bibliography used for the documentation and rules

  ***langcode*.rules** Rules to translate the language, as defined below

  ***langcode*.verify** Sample word to ipa translations

Directory

The domain is AD, the user name is your non-gmail brown user name, and for mac you should use "/" whereas for Windows you should use ""

`\\files.brown.edu\research\CLPS_CohenPriva_Lab\XPF`

`//files.brown.edu/research/CLPS_CohenPriva_Lab/XPF`

Rules

`translate04.py` represents the current stage of how rules should be written. To test your rules you can use `python3` (which could be called `python` if that's what you installed), the name of the script (`translate04.py`), `-l` *rules.file*, *word1*, *word2*, *word3*, where *rules.file* is a csv file with the columns `type`, `sfrom`, `sto`, `weight`, `precede`, `follow`, and `comment`, as in the following example

```
#
# Lines starting with # are ignored
#
type,sfrom,sto,weight,precede,follow,comment
class,front,e|i,,,,
class,passthrough,[abdefiklmnopstu],,,,
sub,k,k,1,,,
sub,c,s,1,,{front},
sub,c,k,0.5,,,
sub,g,g,1,,,
sub,c,tʃ,2,,h,
sub,h,,2,c,,
sub,({passthrough}),\1,0.1,,,
ipasub,\b(.*)\b \1,\1:,1,,,
word,1,u n o,,,,
```

`translate04.py` accepts several parameters to make this process easier:

-l specifies the rules file (as described above), which should be in csv format. #-initial lines are ignored.

-v specifies the log level (higher ⟶ more information)

-c specifies the verification file, which should be in csv or tsv format. #-initial lines are ignored. The first column should contain the word, and the second column its translation. For example:

```
# Test ch rule
che,tʃ e
# Test c[ie] rule
ci,s i
ce,s e
```

-r specifies a file (`-` for standard input), from which words should be read for translation. Only the first word in every line is read, to facilitate a use case with a counts column.

word rules

Word rules are plain substitution rules. If a word matches the "sfrom" column, it is replaced by the *sto* column, and avoids further processing.

pre rules

Preprocessing (*pre*) rules are translation tables from alphabets to alphabets, and apply before `.lower()`. They are meant to catch a number of cases in which lowercase processing may go wrong, e.g. Turkish uppercase *I*, which should lowerase to dotless *ı*. *sfrom* and *sto* have to be equal-sized strings, such that every character in *sfrom* would be translated to the identical-position character in *sto*.

```
type,sfrom,sto,...
pre,"Iİ","ıi"
```

Theoretically, it could be used to collapse meaningless distinctions, e.g. between various kinds of middle-dots, as bellow, but those should be dealt with using character classes, unless that system breaks for some reason.

```
pre,··□□□,·····
```

match rules

Match rules are plain substitution rules at the level of an individual letter. If a letter matches the "sfrom" column, it is replaced by the *sto* column, and avoids further processing (in particular, *sub* rules will not be checked).

sub rules

Sub rules are the core part of the translation scheme. A rule applies if *sfrom* matches the letter, *precede* matches the preceding context, *follow* matches the following context, and the *weight* is the highest of all matching rules (the behavior is not defined if multiple rules with identical weights match). If no rule matches, the letter would be translated to some non-IPA default, currently `@`. Every property (except weight) can have *class* rules.

sfrom represents **one** alphabet letter (as defined by the utf8 standard)

sto zero or more phonemes (space-separated) that the letter should translate to. The phonemes should be in IPA.

precede A regular expression that must match the preceding context. This expression ends with `$` (added by the program), to make sure that the expression would match the context that immediately precedes *sfrom*. `^` in *precede* would therefore designate "beginning of word," as `^$` is an empty string.

follow Similar to *precede*, but for the following context, and starts with `^`, to make sure it applies to the context immediately following the letter in question. `$` would therefore designate "end of word," as "'`^$`' is an empty string.

weight the relative priority of the rule (higher values will override lower values). The goal is to designate rules that take priority over other rules. If we have a default rule for `c` (translate to k), and a specific rule for `c` (translate to tʃ when `h` follows), we want to make sure the specific rule would apply rather than the lower priority rule. The following is an example

```
type,sfrom,sto,weight,precede,follow,comment
sub,c,k,1,,,"Default rule for c"
sub,c,tʃ,2,,h,"Rule should apply rather than the default rule"
```

class rules

Class rules define internal substitution guidelines meant to improve readability or allow reuse to a recurring element. As with word rules, only the *sfrom* and *sto* columns are used. *sfrom* defines the name of the class, and *sto* the value of the class. The following shows how to make a non-latin alphabet more readable in the sub rule section.

```
type,sfrom,sto,...
class,pi,π,
sub,{pi},p,...
```

The following is an example of reuse

```
type,sfrom,sto,precede,follow,...
class,vowels,[aeuio],
sub,c,s,,{vowels},...
sub,g,x,,{vowels},...
```

ipasub rules

ipasub rules perform substitutions on the output IPA directly, and should be used sparingly. *sfrom* designates the input pattern, *sto* the replacement string, and *weight* the order of application, from heaviest to lightest. These can be as complex as python3 allows, but should ideally be used to change what constitutes a phoneme, as in the following cases

```
type,sfrom,sto,...,comment
ipasub,({consonant}) \1,\1 ː,...,double consonants are phoneme + length
ipasub,({vowel}) \1,\1ː,...,double vowels are distinct from singletons (are different phonemes)
ipasub, ʷ,ʷ,...,labialized elements are distinct phonemes (kʷ and k are not the same)
    ipasub,ʲ, ʲ,...,palatalization is not phonemic (but not allophonic either)
```

Utility programs

`stopatn.sh` *n* [*minfreq*]

`stopatn.sh` (stop-a-n) reads word frequency files from standard input (read: that are being piped to it, e.g. using `bzcat es.bz2 | ./stopatn.sh 1000 2`. The input is assumed to have the Crubadan format, i.e. *word freq*. The program removes the any words whose frequency is lower than *minfreq* (defaults to `1` – no filtering), finds the frequency of the *n*th word in decreasing frequency order, and provides the all the words whose frequency is at least as high. The program therefore replaces the use of `head -n` *n*, which introduced an alphabetic bias (if there are multiple words at the *n*th position.

For instance, assuming that the input is:

```
A 5
B 4
C 4
D 3
```

`head -n 2` would provide `A 5` and `B 4`, but `sumstats.sh 2` would also provide `C 4`, because it has the same frequency as `B 4`.

python3 sumstats01.py -l LANGRULES [-c CHECK] [-r READ] [-m MIN] [-N] [-A] [-@ MAX@]

Provide summary statistics for language and frequency files

The parameters that are shared with `translate04.py` have the same syntax, except `-r` defaults to standard input (piped output from `cat`, `bzcat`, or `stopatn.sh`, and to list word frequencies (`translate04.py` uses -r only if specified, and ignores everything except the first word in each line).

-l language code rules file (*.rules* files)

-c verification file (*.verify* files)

-r input to summary statistics evaluation, assumed to have Crubadan word frequency format (*word freq*). Defaults to piped input (stdin).

-m minimum frequency to consider (functionality provided also by `stopatn.sh`). Defaults to 1 (no filtering)

-N suppress summary information

-A Enumerate all words and probabilities (currently not working)

-@ number of @ words to include in summary

Lenition

Consonant lenition processes are a range of phonological processes that tend to result in shorter duration and higher intensity, and are more likely to occur in fast or casual speech, between vowels (or non-nasal sonorants; sometimes a single vowel suffices), and in low-information (frequent, predictable) contexts. These include:

Degemination A long consonant becomes shorter, e.g. tː⟶t (Classical Hebrew *dov* (sg.) vs. dubːim – degemination and spirantization)

Spirantization A stop becomes a fricative, e.g. t⟶θ (as in *water* vs. *Waßer*)

Voicing A voiceless obstruent becomes voiced, e.g. t⟶d (as in *of* vs. *off*)

Debuccalization Loss of place of articulation, e.g. t⟶ʔ, s⟶h (as in button, Cockney English *water*)

Tapping t⟶ɾ (as in American English *water*)

Approximantization t⟶l (as in *was* vs. *were*, *corpus* vs. *corpora*)

Gliding c⟶j (as in *day* vs. *Tag*)

Deletion t⟶0 (as in *jus'*)

For each of the languages you process, you are supposed to document any existing lenition processes in a designated section in the language documentation file.