Effect of Phonological Proximity on the Identification of Novel Languages

By: Zion Smith
12835722

Supervisor: Dr. Marijn van ‘t Veer

BA Thesis Linguistics

UNIVERSITEIT VAN AMSTERDAM

27 June 2022

Zion Smith
Marijn van ‘t Veer
Bachelor’s Thesis

Table of Contents **2**

1. Abstract

Previous research has indicated that languages are often confused with others that are from the same geographic region, yet the underlying mechanism behind this correlation is not clear (Skirgård et al., 2017). In line with the Genus-Macroarea method of language sampling (Miestamo, 2008), I propose that phonological proximity to a hypothetical prototype language of a macroearea is the main factor influencing this perceptual geographic grouping of languages. In order to test this, I created a Python script that aggregates data from two existing language databases in order to compile a prototypical phonology for 10 macroareas, and then generates a series of random syllables adhering to that phonology. Participants were then asked to complete a language identification task using recordings of the prototype languages. Results from the experiment were largely insignificant. However, I was able to demonstrate that participants’ performance varied dramatically and significantly between macroareas. Furthermore, this investigation’s innovative method opens up many opportunities for further research.

2. Introduction

2.1. Background

Globalization has brought Earth’s many languages into closer contact than ever. People today are far more likely to speak a language from an entirely different part of the world, or at the very least, have extensive contact with native speakers of foreign languages. This is especially true in countries with significant immigrant populations, such as the United States. As a result of this, it is also more likely than ever for us to come into contact with a completely novel language, whether in person or over the internet. In this situation we cannot help but try to identify what it is we are listening to. Oftentimes we are able to take a guess as to where the language originates, perhaps because it sounds similar to languages we are already familiar with. On other occasions, we may be left dumbfounded.

Yet as it stands, there is a considerable gap in research relating to the identification of previously unheard languages and the factors that influence such decision-making. In particular, no study has investigated the ability of an individual to correlate a completely novel language to a familiar geographic region. I predict that American adults possess a significant ability to categorize unknown languages by geographic area. Furthermore, I propose that phonological proximity to a hypothetical prototype language should be the most important factor in having listeners associate a language with a region.

Of great use to this study is an analysis of the data from the now-defunct Great Language Game (Skirgård et al., 2017). This online quiz presented players with a short audio recording from an unspecified language, after which players were asked to make a forced-choice guess about the identity of the language. As the game progressed, more choices were added out of the game's pool of 78 languages. The study in question sought to find out which languages were confused for each other, and if any clear patterns could be identified. 15 million guesses were examined. A Neighbor-Net was used to visualize how often each language was confused for each other language in the dataset. Using this, the researchers were able to demonstrate that geographic proximity was by far the best predictor of which languages would be confused with each other. This even proved to be a stronger predictor than genetic relationships, although the two often coincide. For example, the two languages confused with each other most often were Punjabi and Kannada; two languages from completely distinct language families, but with a close geographic proximity and an extensive historical relationship. Additionally, similar phoneme inventories were found to have a significant effect on the likelihood of two languages being confused for each other, although this effect was often overshadowed by geographic proximity.

Dryer (1989) introduces the concept of continent-sized linguistic areas as an explanation for such areal categories that seemingly supersede genetic relationships. He purports that large-scale areal phenomena are more widespread than is generally thought, thanks to ancient language contact or perhaps even deep genetic relationships that go beyond the limits of contemporary linguistics. Miestamo (2008) builds upon this idea with the Genus-Macroarea method of language sampling. In the method, the primary genealogical stratification is made at the genus level, and the primary areal stratification at the level of *macroareas* (Miestamo et al., 2016, p. 247). The findings of Skirgård et al. (2017) seem to show that people categorize languages more by macroarea than by genetic relationship. These perceptual macroareas, however, do not align with those defined in Dryer (1992): Africa, Eurasia, Southeast Asia & Oceania, Australia & New Guinea, North America, and South America. Eurasia in particular does not appear to be a cohesive category among players of the game, as there is a clear division between European languages and Asian languages. Additionally, there is a clear divide between East Asian and South Asian languages (Skirgård et al., 2017, pp. 16).

As interesting as the findings from Skirgård et al. (2017) are, the study was inevitably held back by a number of issues. For the most part, this is because the Great Language Game was never intended to be used for linguistic research. Although it wound up providing the researchers with a truly massive amount of data, it also meant that they were unable to control the means of

data collection. Due to the anonymous nature of the game, the only participant data available was their country of origin, as determined by their IP address. Participants' full language background, level of education, experience in the field of linguistics, and even age were completely unknown. The stimuli used in the game were hardly representative of the world's languages; roughly half of them were from the Indo-European language family, with several continents (North America, South America, Australia) being excluded from the game entirely. Most crucially of all, the researchers had no way to know for sure what participants were basing their guesses on, whether that be phonology, vocal quality, lexical items, or previous experience with the language in question. Nonetheless, the results of the analysis provide an excellent starting point for further research.

Other studies have attempted to specifically examine the phonological stereotypes people hold about particular geographic regions, albeit on a smaller scale. Mitchell et al., (2017) found that African Americans living in a largely working class neighborhood of Columbus, Ohio were able to identify on a physical map of the United States where various dialects of African American English (AAE) could be found. They were even able to indicate where specific phonological features were prominent, and several participants even drew rough isoglosses on the map. For the most part, the stereotypes that participants shared lined up with existing research on the phonological variation of AAE. These findings are crucial in that they show people's ability to correlate (stereotypes about) phonological features to geographic regions. However, the study only explores one community's ability to differentiate dialects of a singular ethnolect, and furthermore only those who belong to its associated ethnicity. It is not made known, for example, whether Americans of any race hold salient phonological stereotypes of languages other than English within the United States, or about languages from any other area of the world.

Over a decade prior, Thomas & Reaser (2004) found that both European Americans and African Americans both had difficulty identifying the ethnicity of African Americans from Hyde County, North Carolina, whose variety differs from "prototypical" AAE because it shares several phonological features with the local European American vernacular. This held true even in the presence of lexical and morphological features emblematic of AAE. It would seem that phonological proximity supersedes other identifying factors in situations where they conflict. Although this study does not deal with geography whatsoever, it is not inconceivable to imagine similar findings in that regard. For example, a language originating from the Middle East may be conflated for a European language if it exhibits a phonology that is more prototypical of Europe than it is of the MENA region. Indeed, Skirgård et al. (2017) find some examples; Hebrew is

confused for European languages (especially Yiddish) far more than it is confused for its closest geographic neighbors and genetic relatives.

At first glance, it may seem that my proposal contradicts the findings of Skirgård et al. (2017). Rather, I believe it supplements them. Claiming that phonological similarity is the strongest factor influencing language identification does not undermine the impact of geographic proximity and historical relationships. Quite the contrary, they are the strongest predictors precisely because they have the largest effect on a language's phonology. Yet in the case of natural languages, it seems that these factors are hopelessly intertwined. As in Skirgård et al. (2017), there seems to be little way to ensure that a participant is solely basing their decision on phonology and not on other linguistic or extralinguistic factors. Additionally, it is nigh impossible that any natural language will exhibit a perfectly prototypical phonology for its region, though it may come fairly close.

This study will attempt to resolve these issues by employing a series of artificially constructed languages. Each language will exhibit a phoneme inventory, syllable structure, and prosody that is prototypical of the linguistic macroarea it represents. These constructed languages will henceforth be referred to as *prototypes*. This all will be accomplished via a Python script that gathers data from two existing language databases: the World Atlas of Language Structures (Dryer & Haspelmath, 2013) and PHOIBLE 2.0 (Moran & McCloy, 2019).

In doing so, this study seeks to answer the following research questions:

1. Are American adults able to identify the origin of an unknown language from a given linguistic macroarea, given that the language exhibits a phonology prototypical of that macroarea?
2. How does this ability vary between macroareas?

This ambitious project calls for a highly innovative method of data collection and stimuli generation.

2.2. Phonological Macrosampling Via Existing Language Repositories

For the purposes of this investigation, we have the opportunity to experiment with our data sampling method. In most cases, linguists are required to carefully select a diverse and representative small sample of languages that fits the scope of their research project. However, by using the existing language databases of PHOIBLE 2.0 and the World Atlas of Language Structures (WALS), we can forgo the sampling process entirely and analyze every single

language present in the database within a specified set of geographic boundaries. “Macrosampling” in this manner is only feasible due to the automated manner of stimulus generation employed. Such a large sample of data should increase the accuracy of the script’s calculations significantly.

PHOIBLE 2.0, or the Phonetics Information Base and Lexicon, is a repository of cross-linguistic phonological inventory data, which have been extracted from source documents and tertiary databases and compiled into a single searchable convenience sample (Moran et al., 2019). Release 2.0 from 2019 includes 3,020 inventories that contain 3,183 segment types found in 2,186 distinct languages. This database will serve as the basis for generating a segment inventory for a prototype language that is representative of a given macroarea.

WALS (Dryer & Haspelmath, 2013) is a large database of structural (phonological, grammatical, lexical) properties of languages gathered from descriptive materials (such as reference grammars) by a team of 55 authors. There are 144 chapters in the database, of which this study will use just 2. The first is Chapter 12: Syllable Structure (Maddieson, 2013) which compares primarily the maximal syllable from a large sample of world languages. The second is Chapter 17: Rhythm Type which examines the presence or absence of rhythmic stress, also from a large sample of world languages. These two chapters will allow us to determine important phonological features for our generated prototype languages, aside from segment inventory.

Although this study adopts some terminology from the Genus-Macroarea method of language sampling, and many of its assumptions are based on that underlying theory, it will not employ the method itself. Every language present in the databases will be used by the algorithm, regardless of its family or genus. For the purposes of this study, to do otherwise would be unnecessary and perhaps even counter-productive. The goal is not to prove any underlying phonological tendencies as occurring independently of genetic relationships. Rather, we are only concerned with the surface-level patterns average people will perceive. Overrepresentation of minority language families in the algorithm will likely yield patterns that are unrecognizable to the average person. It could also be argued that languages should be weighted according to their total number of speakers. However, this is bound to pose problems in some macroareas. Mandarin Chinese, for example, has such a large speaking population in East Asia that it would render all other languages virtually irrelevant, spare perhaps Japanese. For the purposes of this study, the simplest solution is to weigh every language equally.

3. Methods

3.1. Construction of Prototype Phonology

A series of artificially constructed languages, or prototypes, was created using a Python script that incorporates the existing language databases of WALS and PHOIBLE 2.0. These prototypes are devoid of meaningful lexical items, morphology, and syntax, but appear outwardly to be as complex as a natural language. Each prototype has a phoneme inventory, syllable structure, and prosody that is prototypical of the linguistic macroarea it is meant to represent. Each factor is explained in detail below.

Consonants

The consonant inventory was determined using PHOIBLE 2.0. A consonantal place of articulation, manner of articulation, or manner of voicing was added to the constructed language's phonology if it occurred in at least 60% of the specified macroarea's languoids. For example: despite being common cross-linguistically, fricatives occurred in only a small minority of Australian languages, so they were not added to the prototype's phonology. On the other hand, the cross-linguistically rare lamino-alveolar place of articulation occurred in over 60% of Australian languages, so it was added to the prototype's phonology.

Out of the natural languages that had each category, the average number of segments occupying that category was determined, and that number of the most common segments was retrieved and added to the prototype's phonology. For example: out of the languages in the Middle East & North Africa macroarea that have velar consonants, the average inventory of velars was determined to be 4. Thus, the 4 most frequently occurring velars were identified; [k], [g], [x], and [ɣ]. These segments were subsequently added to the prototype's phonology.

After each category had been filled, the consonant inventory was pared down by removing every segment that did not appear in at least one place of articulation, manner of articulation, and manner of voicing in the prototype's phonology. In the Australian prototype phonology, [ʈ] was one of the most frequently occurring retroflex consonants. However, it was not one of the most frequently occurring plosives, and it was therefore removed from the phonology.

Vowels

The vowel inventory was determined using PHOIBLE 2.0. It was constructed in much the same way as the consonant inventory, but with the relevant categories being vowel type and vowel length. Valid vowel types included voiced, voiceless, and nasal. Vowel length was measured on a

simple short or long binary. During the process of stimuli generation, all vowels in a prototype's phonology were equally likely to occur.

Tones

The tone inventory was determined using PHOIBLE 2.0. Again, this was done in much the same way as the consonant and vowel inventories. However, tone was not split into any categories whatsoever. Tone as a whole was included in a prototype's phonology if it was present in at least 60% of the languages in that macroarea. The average tone inventory size was then calculated out of that set of languages, and that number of the most commonly occurring tone segments was added to the prototype's phonology. As can be seen in subsection 3.4, tone was only present in the Sub-Saharan African prototype.

Syllable Structure

The syllable structure was determined using WALS Chapter 12. The most frequently occurring syllable structure within a macroarea was added to the constructed language's phonology. The possible values, as determined by WALS, were Simple, Moderately Complex, and Complex. Simple is defined as having a maximal syllable of CV. Moderately Complex is defined as having a maximal syllable of CCV or CVC. Languages with a Complex syllable structure are those with any maximal syllable greater than CCV or CVC. For the purposes of this experiment, any prototype with a Complex syllable structure was treated as having a maximal syllable of CCVCC.

Rhythm Type

The rhythm type was determined using WALS Chapter 17. As with syllable structure, the most frequently occurring rhythm type was added to the prototype's phonology. The possible values, once again determined by WALS, were Trochaic, Iambic, Dual, or None. For the purposes of this experiment, constructed languages with a rhythm type of Dual or None were treated identically, and stress was placed randomly on either the left-hand or right-hand syllable in disyllabic words.

Emblematic Phonemes

The final step of prototype phonology generation was the identification of emblematic phonemes. A segment was considered emblematic if it occurred in, at most, just 1 other prototype phonology. Phonemes that were identified as emblematic were used more frequently in stimuli generation.

3.2. Generation of Stimuli

After a full set of phonological parameters had been produced, a second script was used to generate a string of 40 syllables adhering to each prototype’s phonology. In addition to the criteria unique to each language, all syllables were made to strictly adhere to the sonority hierarchy, and every word was either monosyllabic or disyllabic. Every segment in the language’s phoneme inventory appears at least once in its string. Additionally, the script identifies a number of emblematic phonemes for each language which appear in, at most, one other prototype. These emblematic phonemes appear at least twice in each sample.

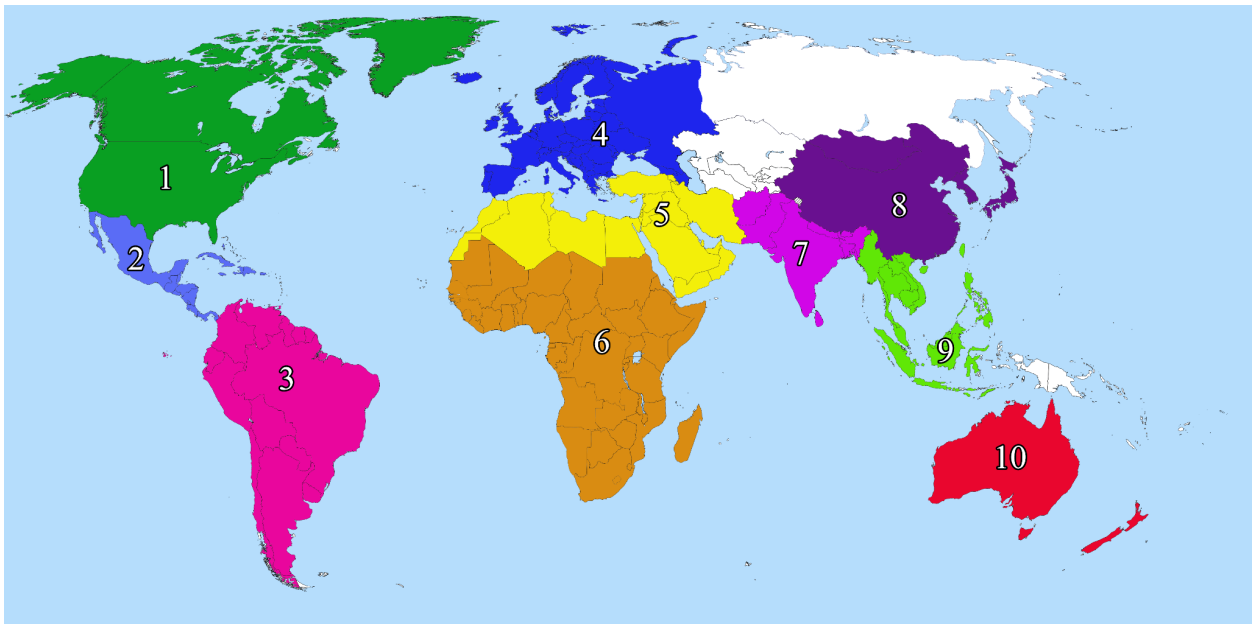
The full scripts for both phonology construction and stimuli generation may be found in Appendix C.

3.3. Macroarea Parameters

The boundaries of each macroarea used can be seen in Figure 1, which was also presented to participants. Rather than using the traditional six macroareas as defined by Dryer (1992), several have been divided into smaller blocks. Influenced by the results of Skirgård et al. (2017), Eurasia has been divided into Europe, Central Asia, South Asia, East Asia, and Southeast Asia. In the interest of maintaining a comparable size between macroareas, the Middle East and North Africa (MENA) region has been separated from Sub-Saharan Africa, and Central America has been separated from the rest of North America. Oceanic languages have been excluded from the experiment entirely, as it would be difficult to easily convey their geographic boundaries to participants. Much of North and Central Asia was excluded from the experiment as well. One could argue that this region contains a notable sprachbund (Altaic), but it was assumed that such a macroarea would not be recognizable to participants. Finally, the island of Papua was not included. Despite having an exceptional amount of linguistic diversity, it was believed that participants would consider it too small to constitute its own macroarea.

The macroareas are also listed in Appendix A, alongside their latitudinal and longitudinal parameters that served as the script’s initial input.

Figure 1 - World Map, Numbered by Macroarea



1. North America
2. Central America
3. South America
4. Europe
5. Middle East & North Africa (MENA)
6. Sub-Saharan Africa
7. India
8. East Asia
9. Southeast Asia
10. Australia

3.4. Prototype Phonologies

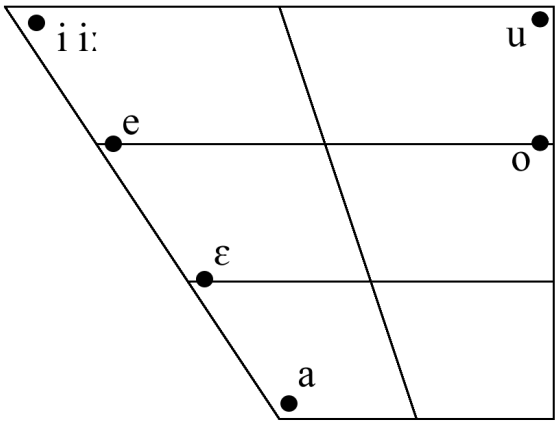
Overall, the script was very successful in producing phonologies that were both believably structured and distinct from each other. However, 2 out of the 10 macroareas exhibited no emblematic phonemes whatsoever. These macroareas may still be identified by their segment inventory as a whole, as well as by their syllable structure and rhythm type. The full phonological parameters for each prototype language may be found in the following pages.

1. NORTH AMERICA

Table 1 - North America Consonants

	Bilabial	Alveolar	Postalveolar	Palatal	Velar	Uvular	Glottal
Plosive	p b	t t'			k k' k^w	q q'	
Nasal	m	n					
Fricative		s	ʃ		x	χ	h
Affricate		ts	t͡ʃ				
Approximant	(w)			j	(w)		
Lateral Approximant		l					

Figure 2 - North America Vowels



Consonants: 21
Vowels: 7

Syllable Structure: Complex
Rhythm: None
Emblematic Phonemes: [k'], [k^w], [t'], [χ]

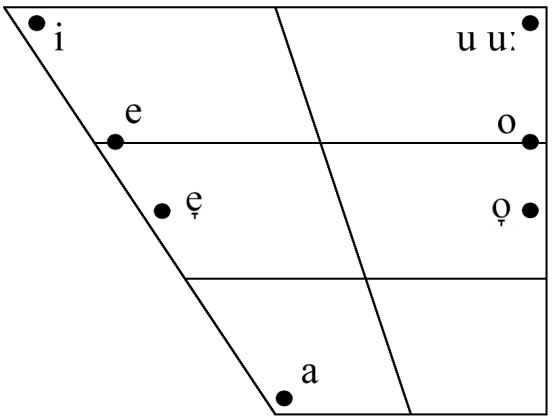
North America's phonology is noteworthy due to the presence of uvular and ejective consonants, as well its complex syllable structure. Additionally, it has a vowel inventory that is quite small.

2. CENTRAL AMERICA

Table 2 - Central America Consonants

	Bilabial	Alveolar	Postalveolar	Palatal	Velar	Glottal
Plosive	p b	t			k g	
Nasal	m	n				
Fricative		s	ʃ			h
Affricate		ts	t̠ʃ			
Approximant	(w)			j	(w)	
Lateral Approximant		l				

Figure 3 - Central America Vowels



Consonants: 15
Vowels: 8

Syllable Structure: Complex
Rhythm: None
Emblematic Phonemes: None

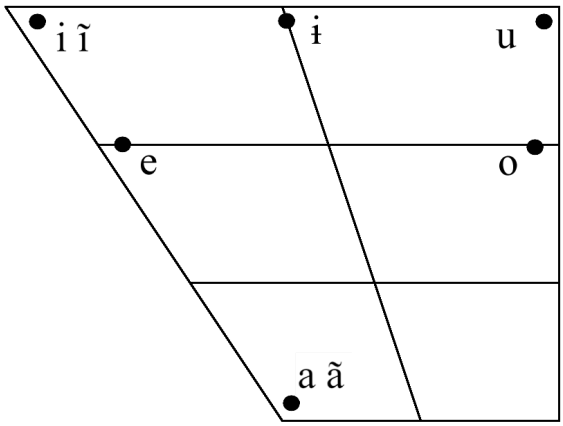
Central America’s phonology is one of only two with no emblematic phonemes whatsoever. Overall, it has very few identifiably unique features per the criteria of this study. It has a consonant inventory that is slightly smaller than average.

3. SOUTH AMERICA

Table 3 - South America Consonants

	Bilabial	Alveolar	Postalveolar	Velar	Glottal
Plosive	p b	t d		k	
Nasal	m	n			
Tap/Flap		ɾ			
Fricative		s			h
Affricate			t͡ʃ		
Approximant	(w)			(w)	

Figure 4 - South America Vowels



Consonants: 12
Vowels: 8

Syllable Structure: Moderately Complex
Rhythm: Trochaic
Emblematic Phonemes: [ɨ]', [ã]

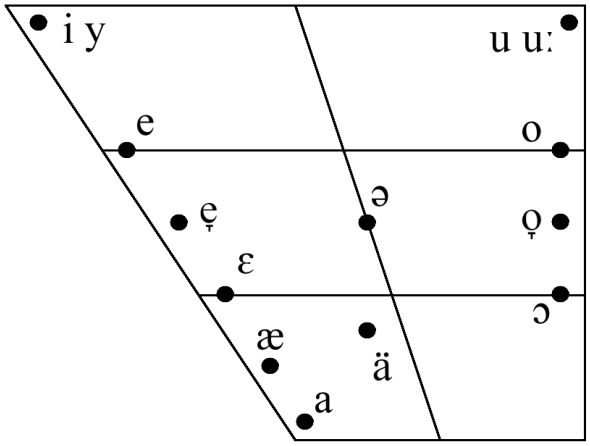
South America’s phonology is notable for its considerably small consonant inventory, which is also the only inventory to contain the alveolar tap. It is one of just two prototypes to feature nasal vowels. It exhibits a strictly trochaic rhythm.

4. EUROPE

Table 4 - Europe Consonants

	Bilabial	Labiodental	Dental	Alveolar	Postalveolar	Palatal	Velar	Glottal
Plosive	p b		t̪ d̪	t d		c	k g	
Nasal	m			n		ɲ	ŋ	
Trill				r				
Fricative		f v	s̪	s z	ʃ ʒ		x	h
Affricate				ts dz	t̠ʃ			
Approximant						j		
Lateral Approximant				l		ʎ		

Figure 5 - Europe Vowels



Consonants: 29
Vowels: 14

Syllable Structure: Complex
Rhythm: Trochaic
Emblematic Phonemes: [ʒ], [s̪], [dz], [ʎ], [æ], [ä]

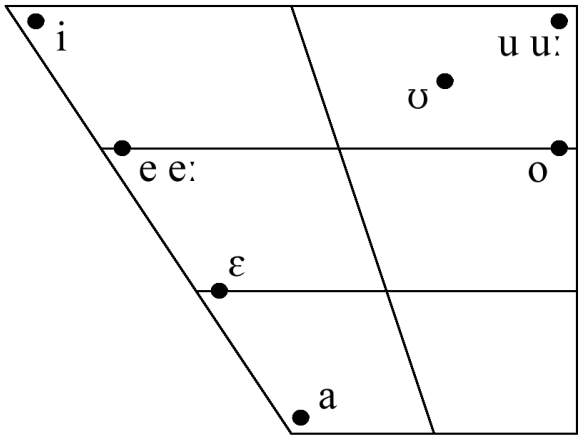
Europe’s phonology has both the largest consonant and vowel inventory. It also has the most emblematic phonemes of any region. It is also notable for having extensive voicing contrasts, several palatal consonants, and front-rounded vowels. Europe has a strictly trochaic rhythm type and complex syllable structure.

5. Middle East & North Africa

Table 5 - MENA Consonants

	Bilabial	Labiodental	Dental	Alveolar	Postalveolar	Palatal	Velar	Uvular	Pharyngeal	Glottal
Plosive	p b		t̪ d̪	t d			k g	q		
Nasal	m		n̪	n						
Trill				r						
Fricative		f v		s z	ʃ		x ɣ		ħ	h
Affricate					t̪ʃ d̪ʒ					
Approximant	(w)					j	(w)			
Lateral Approximant				l						

Figure 6 - MENA Vowels



Consonants: 27
Vowels: 9

Syllable Structure: Moderately Complex
Rhythm: Trochaic
Emblematic Phonemes: [ħ], [ɣ]

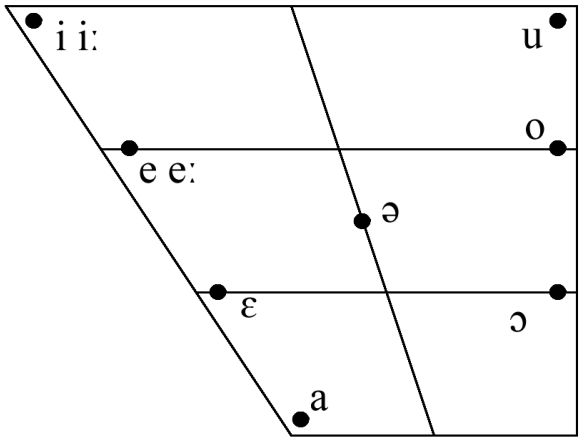
The Middle East and North Africa region is noteworthy for the presence of uvular plosives and pharyngeal fricatives. It has a large consonant inventory, and fairly average vowel inventory. Its rhythm is strictly trochaic.

6. SUB-SAHARAN AFRICA

Table 6 - Sub-Saharan African Consonants

	Bilabial	Labiodental	Alveolar	Postalveolar	Palatal	Velar	Uvular	Glottal
Plosive	p b (kp) (gb)		t d			k g (kp) (gb)		
Implosive	ɓ							
Nasal	m		n		ɲ			
Fricative		f v	s z					h
Affricate				t͡ʃ d͡ʒ				
Approximant	(w)				j	(w)		
Lateral Approximant			l					

Figure 7 - Sub-Saharan African Vowels



Consonants: 22
Vowels: 10

Tones: ˥, ˧, ˦
Syllable Structure: Moderately Complex
Rhythm: Trochaic
Emblematic Phonemes: kp, gb, ɓ

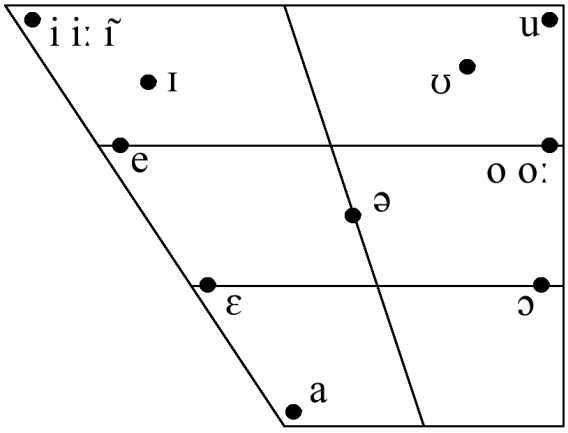
The Sub-Saharan Africa prototype is the only one to feature phonemic tone, of which it has a simple high, mid, and low contrast. It is also noteworthy for the presence of labial-velar plosives as well as the bilabial implosive. Its rhythm type is strictly trochaic.

7. INDIA

Table 7 - Indian Consonants

	Bilabial	Dental	Alveolar	Postalveolar	Retroflex	Palatal	Velar	Glottal
Plosive	p pʰ b	t̪ d̪	t d		ʈ ɖ		k kʰ g	
Nasal	m		n			ɲ	ŋ	
Trill			r					
Fricative			s z					h
Affricate						cç, ɟʝ		
Approximant						j		
Lateral Approximant			l					

Figure 8 - Indian Vowels



Consonants: 24
Vowels: 13

Syllable Structure: Moderately Complex
Rhythm: None
Emblematic Phonemes: ʈ, ɖ, cç, ɟʝ, ɪ

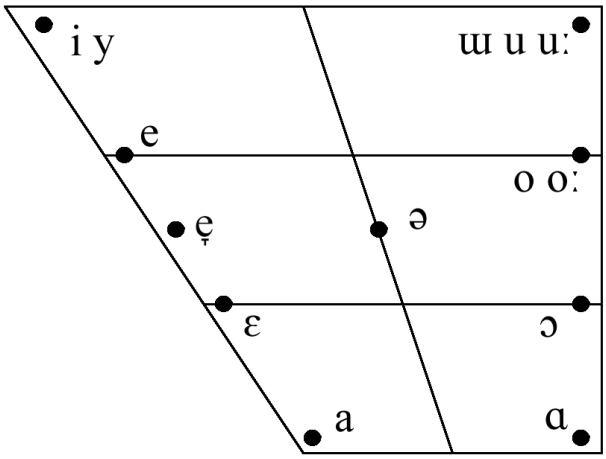
The Indian prototype is one of only two to feature retroflex consonants, and is the only region with palatal affricates. It also has three-way voicing contrasts for the bilabial and velar plosives. Along with South America, it is one of two prototypes to contain nasal vowels. It has a fairly large consonant and vowel inventory.

8. EAST ASIA

Table 8 - East Asian Consonants

	Bilabial	Labiodental	Dental	Alveolar	Postalveolar	Palatal	Velar	Glottal
Plosive	p pʰ b		t̪	t d			k kʰ g	
Nasal	m			n		ɲ	ŋ	
Trill				r				
Fricative		f		s z	ʃ			h
Affricate				ts tsʰ	ʧ ʧʰ			
Approximant	(w)					j	(w)	
Lateral Approximant				l				

Figure 9 - East Asian Vowels



Consonants: 27
Vowels: 14

Syllable Structure: Moderately Complex
Rhythm: None
Emblematic Phonemes: [tʰ], [ʧʰ], [ɑ]

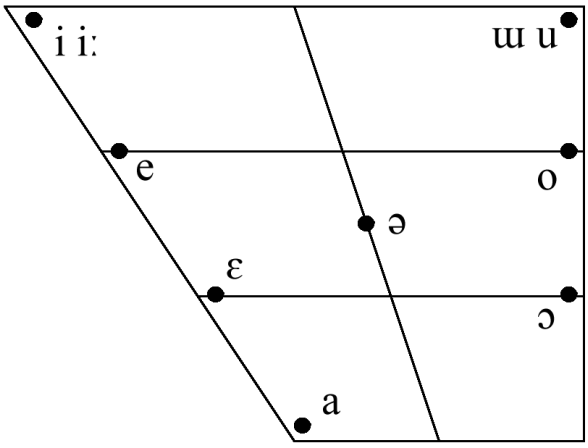
East Asia has three way voicing contrasts with its bilabial and velar plosives, similar to India. It is also noteworthy for its aspirated affricates, front rounded vowels, and unrounded back vowels. Interestingly, the East Asian prototype does not contain phonemic tone. It has a fairly large consonant and vowel inventory.

9. SOUTHEAST ASIA

Table 9 - Southeast Asian Consonants

	Bilabial	Alveolar	Palatal	Velar	Uvular	Glottal
Plosive	p p^h b	t d		k g		
Nasal	m	n	ɲ	ŋ		
Fricative		s				h
Approximant	(w)			(w)		
Lateral Approximant		l				

Figure 10 - Southeast Asian Vowels



Consonants: 15
Vowels: 10

Syllable Structure: Moderately Complex
Rhythm: Trochaic
Emblematic Phonemes: None

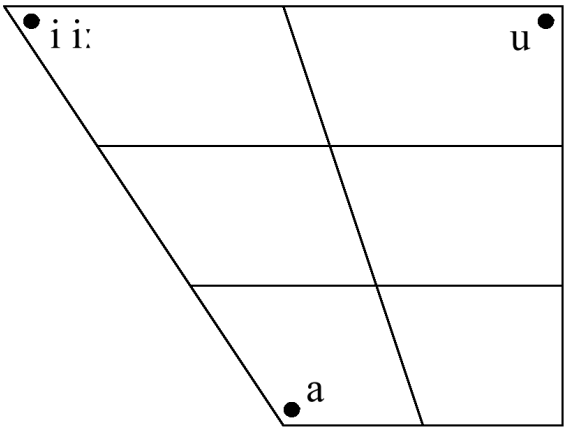
Southeast Asia is the second region with no emblematic phonemes. However, it can be identified by the presence of a three-way voicing contrast in the bilabial plosives, an extensive inventory of nasals, and an otherwise quite small consonant inventory. Its rhythm is strictly trochaic.

10. AUSTRALIA

Table 10 - Australian Consonants

	Bilabial	Dental	Alveolar	Lamino-alveolar	Retroflex	Palatal	Velar
Plosive	p	t̪	t	ʈ			k
Nasal	m		n	ɳ		ɲ	ŋ
Approximant	(w)					j	(w)
Lateral Approximant			l		ɭ		

Figure 11 - Australian Vowels



Consonants: 14
Vowels: 4

Syllable Structure: Moderately Complex
Rhythm: Trochaic
Emblematic Phonemes: [ɳ], [ʈ], [ɭ]

Australia can be identified by its numerous places of articulation, including retroflex and lamino-alveolar consonants. On the other hand, it has an exceptionally small number of manners of articulation. Australia is the only region to completely lack fricatives. It also has the smallest vowel inventory of any region. Its rhythm is strictly trochaic.

3.5. Generated Strings

Once fully generated, each sequence of 40 syllables was split into two halves, in order to create a total of 20 strings consisting of 20 syllables each.

Transcripts of the 20 produced strings can be found in Appendix A.

Each string of syllables was performed orally by one researcher. Said researcher is well-trained in phonology and transcription, and attempted to pronounce each string as accurately as possible. The speaker was allowed to make minute changes to the script during recording. For example, in the case of a consonant cluster that was deemed too difficult or impossible to pronounce, the speaker may have chosen to insert a schwa or other reduced vowel.

The researcher responsible for performing the strings did not struggle very much with pronouncing any sequences of phonemes. However, there were a handful of syllables that were difficult to pronounce, as well as some that the researcher deemed impossible. One example is the impossible syllable [ɓmuɬ] which occurred in the Sub-Saharan African string. The researcher opted to remove the nasal [m] entirely in favor of maintaining the emblematic implosive [ɓ]. Every change was made in agreement with two assistants who were monitoring the recording session. These issues are likely the result of phonotactics being largely ignored during the construction of the stimuli. Phonotactics are mentioned in further detail in Section 5. Nonetheless, the two assistants assessed the output of the script as being quite naturalistic.

3.6. Experimental Method

The method of data collection for this study was an online questionnaire. Participation in the study was limited to adults (aged 16 or higher) with American citizenship. Over a period of several weeks, the questionnaire received 82 responses. In the first phase of the questionnaire, participants were presented with one of the 20 stimuli. They were then asked to indicate which geographic region they believe the language originated from. This was a forced-choice selection out of the set of 10 macroareas used in the study. A map was displayed on screen to demonstrate to participants where the boundaries of each region lie. Participants were also asked whether they are familiar with the language in question, and if so, to identify which language it is. This option is included purely to dissuade participants from realizing that the stimuli are not taken from natural languages. Participants completed this guessing task for each of the 20 stimuli.

The second phase of the questionnaire asked participants a number of questions about themselves. They were asked to indicate their nationality, age, level of education, history of living abroad, and history of language exposure. Additionally, each participant was asked to indicate which factors they believe influenced their decisions in the first phase of the experiment.

Following completion of both phases, participants were debriefed about the true nature of the experiment. They were informed that the samples presented to them were not from naturally occurring languages, but were in fact the output of a computer script. Additionally, participants were allowed to relisten to any sample they wished, but this time labeled with the correct answer. Participants were also allowed to inspect their final score.

4. Results

Overall performance was noticeably poor. The mean score for the experiment was a mere 3.77 out of a maximum score of 20, or in other words an accuracy of 18.85%. However, this performance still indicates that participants were guessing at a level higher than chance. Additionally, this performance varied significantly between regions. The macroarea identified correctly most frequently was Europe, with a fairly high accuracy of 56.1%. The macroarea identified correctly the least frequently was South America, with a miniscule accuracy rate of 3.7%. Several other macroareas were also guessed correctly with a rate of roughly 10% or less, the rate to be expected if participants were guessing randomly. These macroareas are South America, Central America, North America, and Australia.

By cross-examining participants' responses about their background with their answers, we can see that having lived in a given macroarea has a varying effect on a participants ability to identify the language from that region (Table 11). It appears that with macroareas which had a high overall accuracy, having lived in that macroarea can increase accuracy even further, as with the MENA stimuli, but it can also have no noticeable effect, as with the Europe stimuli. On the other hand, with macroareas which had a low overall accuracy, having lived in the macroarea can have a negative effect on accuracy. Ultimately though, these tendencies are far from consistent, and none of them are statistically significant overall effect ($p = 0.745$). Similar results can be seen with language experience, likely because the two are often linked (Table X). Once again, language exposure was found to have no significant effect on accuracy ($p = 0.744$).

Table 11 - Accuracy Between Macroareas and Participant Backgrounds

	Overall Accuracy	Lived	Not Lived	Experience	No Experience
North America	0.103659	0.101266	0.166667	0.096154	0.107143
Central America	0.054878	0	0.058442	0	0.059211
South America	0.036585	0	0.037037	0	0.037037
Europe	0.560976	0.551724	0.566038	0.54902	0.580645
MENA	0.292683	0.928571	0.233333	0.928571	0.233333
Subsaharan Africa	0.207317	0.166667	0.208861	0.166667	0.208861
India	0.176829	0	0.179012	0	0.179012
East Asia	0.170732	0.25	0.166667	0.1	0.175325
Southeast Asia	0.207317	0.25	0.20625	0.375	0.198718
Australia	0.073171	0	0.074074	0	0.074074

Other aspects of a participant’s background were also found to have no significant effect on their overall score. This includes age ($p = 0.7365$), educational background ($p = 0.786$), and level of experience in the field of linguistics ($p = 0.975$).

Participants were asked which factors they believed influenced their decision-making during the experiment. The answers to this question can be seen in Table X. As expected, the most frequent options chosen were those that were considered in the construction of the stimuli: consonants and vowels, tones, stress and rhythm, and the complexity of the syllables. Additionally, 23 participants out of 82 indicated that the speaker’s vocal quality influenced their decision-making. As this researcher was a native speaker of English, this may be partially responsible for skewing the responses in favor of Europe.

Table 12 - Influencing Factors Identified by Participants

	Frequency
Consonants & Vowels	71
Tones	63
Stress and/or Rhythm	56
Complexity of the syllables	29
Speaker's vocal quality	23
Familiar Words	14
Talking speed	13
Volume	2
Grammar	1
Other:	4

5. Conclusion and Discussion

5.1. Data Analysis

Given the exceptionally low performance from the participants, it is not suitable to make any overarching conclusions about the effect of phonological proximity on the identification of unknown languages. However, the results also do not disconfirm the hypothesis that phonological proximity is the most relevant factor, as participants were evidently not guessing completely randomly. There are several other explanations for the results. For example, phonological proximity may only act as an exceptional identifying factor alongside some other factors. Due to this study only testing American adults, it may be the case that adults of other nationalities may exhibit a far clearer effect of phonological proximity on language identification. It is also possible that the stimuli were deficient in some way. Potential improvements to the stimulus can be found in subsection 5.2.

As there was no significant effect of age, level of education, or experience in the field of linguistics, it is not possible to make any conclusions about the effect of these variables. Likewise, there was no significant effect of living history or language experience on a participant’s overall accuracy, and it is not possible to identify a correlation there.

What these results do indicate, however, is that American adults are better at identifying languages with prototypical phonologies from certain macroareas than they are from others. In particular, they do decently well at identifying such languages from Europe and the Middle East and North Africa region, and do badly at identifying such languages from the Americas and Australia. What is not discernible from this data is why such a discrepancy exists. There are many possible explanations, however. Although previous language experience was found to not have a significant effect, it is possible that the total population of speakers from each macroarea is a relevant factor. This could partially explain the discrepancy between macroareas. It may also be the case that the total population of speakers is not the most important factor, but rather the representation of a macroarea’s languages in popular culture or education.

Perhaps the presence of a large “emblematic” phoneme inventory is correlated with a higher accuracy rate. This could certainly explain the accuracy at which Europe was guessed, as its prototype has the most emblematic phonemes by far. However, emblematic phonemes were defined somewhat arbitrarily in this present study, and were only defined relative to the other macroareas tested in the study. For this reason, it makes little sense to treat it as a legitimate independent variable.

Another possible explanation is that linguistically diverse macroareas are identified less often than languages from macroareas that are dominated by a small number of language families. If this were the case, it would indicate that phonological proximity is influenced by genetic relationships far more than it is by geographical proximity. It could also help explain some of the results from this study. For example, Europe’s languages overwhelmingly belong to the Indo-European language family, while the Americas are home to an incredibly large set of language families. However, this could not explain Australia’s low rate of identification, as the continent is largely dominated by the Pama-Nyungan family. Of course, this discrepancy between macroareas is likely due to a conflux of numerous factors. In order to identify whether or not these factors truly have an effect, further research is required. Possibilities for future studies are outlined in the next subsection.

5.2. Methodological Improvements and Possibilities for Further Study

In order to answer this investigation’s initial research question about the effect of phonological proximity on language identification, a similar study to this one could be carried out, but with a number of modifications. For example, our present study could have benefited greatly from a proper control stimulus. This could be one constructed language whose phonology is not prototypical of any macroarea, and perhaps one constructed language whose phonology is based on the complete set of all world languages. Additionally, it could be wise to create a continuum of languages for each macroarea which range from 0% to 100% phonological proximity to the theoretical prototype language. In other words, one stimulus could be identical to the prototype used in this study, one stimulus could have a phonology with a completely different set of phonemes, and yet another stimulus could share roughly 50% of the segments with the prototype. Such a setup would better allow for an analysis of phonological proximity, separated from other relevant variables.

The method of phonological macrosampling and automated stimulus generation used in this project was ambitious and very innovative. Despite the inconclusive results of the experiment itself, it demonstrated that macrosampling can be a very useful tool for linguistics research. The sheer amount of data processed in this investigation was only possible using an automated script, and would not have been manageable using more traditional methods. However, it was not necessarily employed perfectly, and several improvements could be made to this experiment’s method. Some factors were not taken into account in the construction of the stimulus that perhaps could have been. One example is word length. It is true that word boundaries are often hard to discern in continuous speech, but the length of a word can affect

other aspects of speech, most notably rhythm. Another important factor that was largely ignored is phonotactics. Phonemes often do not occur equally often in the onset as they do in the coda and vice versa. One example is the velar nasal [ŋ], which is fairly common in Europe, but rarely if ever occurs word-initially (Anderson, 2013). In our current study, this issue was not accounted for whatsoever, and [ŋ] was allowed to occur in the onset position in the European speech sample. Similarly, not all phonemes are equally likely to appear in consonant clusters as others. This issue was partially remedied by having all generated syllables adhere strictly to the sonority hierarchy, but this approach misses the multitude of subtle nuances between the phonotactics of languages around the world.

Both word length and phonotactics were not excluded intentionally, but due to a lack of relevant data; neither WALS nor PHOIBLE 2.0 contain the relevant information. Thus, future versions of the sampling method employed in this study could benefit greatly from incorporating data from more language repositories. At the moment, it appears that such repositories are not readily available or do not yet exist. This study hopefully demonstrates that these repositories are invaluable to linguistic research as is, and that their expansion could open up countless opportunities for future research.

5.3. Closing

This study sought to investigate the effect of phonological proximity on the identification of unknown languages. Using an elaborate Python script, a series of prototype languages were created based on 10 macroareas. These languages were then used as part of a language guessing game played by 82 participants. Overall performance in the guessing task was far lower than expected, making it unclear whether phonological proximity truly is the most important factor in the identification of unknown languages. However, it was demonstrated that participants' performance varied greatly between macroareas. The reason behind this is not immediately clear, and there are many possible explanations. Further studies could improve on the methodology of this experiment to more effectively assess the effect of phonological proximity on language identification, or to assess the underlying reason why some macroareas are identified more easily than others. All things considered, this study served as an effective pilot test for an innovative method of language sampling and stimulus generation, and it has opened up several avenues for further research.

Bibliography

Dryer, M. S. (1992). The Greenbergian word order correlations. Language, 68(1), 81. https://doi.org/10.2307/416370

Dryer, M. S. (1989). Large linguistic areas and language sampling. Studies in Language, 13(2), 257-292. https://doi.org/10.1075/sl.13.2.03dry

Dryer, Matthew S. & Haspelmath, Martin (eds.) 2013. The World Atlas of Language Structures Online. Leipzig: Max Planck Institute for Evolutionary Anthropology. (Available online at http://wals.info, Accessed on 2022-06-23.)

Gregory D.S. Anderson. 2013. The Velar Nasal. In: Dryer, Matthew S. & Haspelmath, Martin (eds.) The World Atlas of Language Structures Online. Leipzig: Max Planck Institute for Evolutionary Anthropology. (Available online at http://wals.info/chapter/9, Accessed on 2022-06-23.)

Ian Maddieson. 2013. Syllable Structure. In: Dryer, Matthew S. & Haspelmath, Martin (eds.) The World Atlas of Language Structures Online. Leipzig: Max Planck Institute for Evolutionary Anthropology. (Available online at http://wals.info/chapter/12, Accessed on 2022-06-23.)

Miestamo, M. (2008). Standard negation. https://doi.org/10.1515/9783110197631

Miestamo, M., Bakker, D., & Arppe, A. (2016). Sampling for variety. Linguistic Typology, 20(2), 233-296. https://doi.org/10.1515/lingty-2016-0006

Mitchell, D., Lesho, M., & Walker, A. (2017). Folk perception of African American English regional variation. Journal of Linguistic Geography, 5(1), 1-16. https://doi.org/10.1017/jlg.2017.2

Moran, Steven & McCloy, Daniel (eds.) 2019. PHOIBLE 2.0. Jena: Max Planck Institute for the Science of Human History. (Available online at http://phoible.org, Accessed on 2022-06-23.)

Rob Goedemans, Harry van der Hulst. 2013. Rhythm Types. In: Dryer, Matthew S. & Haspelmath, Martin (eds.) The World Atlas of Language Structures Online. Leipzig: Max Planck Institute for Evolutionary Anthropology. (Available online at http://wals.info/chapter/17, Accessed on 2022-06-23.)

Skirgård, H., Roberts, S. G., & Yencken, L. (2017). Why are some languages confused for others? Investigating data from the great language game. PLOS ONE, 12(4), e0165934. https://doi.org/10.1371/journal.pone.0165934

Thomas, E. R., & Reaser, J. (2004). Delimiting perceptual cues used for the ethnic labeling of African American and European American voices. Journal of Sociolinguistics, 8(1), 54-87. https://doi.org/10.1111/j.1467-9841.2004.00251.x

Appendix A - Macroarea Parameters

The following table contains the latitude and longitude values inserted into the script to define the 10 tested macroareas. Different combinations of values can generate different geographic regions for testing.

Table A - Macroarea Parameters

Macroarea	Min. Latitude(s)	Max. Latitude(s)	Min. Longitude(s)	Max. Longitudes(s)
1. North America (NA)	32.6, 68, 53, 25, 29	84, 84, 72, 35, 35	-140, -75, -169, -100, -120	-30, -8, -140, -79, -100
2. Central America (CA)	10, 7, 26, 22, 12	26, 11, 29, 32, 25	-120, -86, -104, -120, -86	-83, -76, -98, -104, -60
3. South America (SA)	-57	13	-82	-31
4. Europe (EU)	36, 62, 76, 43	72, 67, 81, 78	-12, -25, 10, 21	28, -11, 33, 60
5. Middle East & North Africa (MENA)	19, 22, 12.5, 16.5	37, 41, 30, 30	-20, 27, 43, 40	39, 63, 60, 43
6. Sub-Saharan Africa (AF)	-37, -6, -28	19, 15.5, -10	-20, 35, 42	40.5, 52, 53
7. India (IN)	4, 25	30, 37	61, 61	97, 80
8. East Asia (EA)	28, 34, 22, 18	52, 42, 28.5, 25	79, 73.5, 98, 108	148, 79, 134, 118
9. Southeast Asia (SEA)	-10, 15, 16, -11, 3	18, 28.5, 23, 8, 21	92, 93, 101, 107, 114	110, 99.5, 107, 130, 128
10. Australia (AU)	-45, -48	-11, -33	111, 165	155, 180

Appendix B - Produced Strings Per Macroarea

North America

1. ˈjiː.ojm. wuj.ˈmiːq. ˈʧɛts.
ˈʧokʷ. wuχ.ˈtsejh. ˈlɛkʼ. ˈxus.
loχ.ˈwaj. ˈʧnitʃ. na.ˈtsli.
ˈni. ˈqiːm. ˈoh. ˈnwi.

2. ˈa. ˈjin. ˈxɛwts. ˈhen.
ˈnotʃ.nol. ˈleb. ˈʧme.tʼu.
jets.ˈtweh. ˈliːntʃ.kʷmɛkʷ. tsjejq.ˈjiːts.
ˈtlɛnk. ˈʃje. ˈlɛ. tʼow.ˈpla.
ˈtsɛkʼ.

Central America

1. ˈʃluh.hǫm. ˈnow.wuː. ˈʃmuk.
os.ˈma. ˈjil. ˈbotʃ.
ˈpa. ˈneǫns. ˈnits.top.
ˈwat. ˈmo. ˈgitʃ. neb.ˈleǫn.
hul.ˈwets. ˈliwts.peǫnt. ˈla.

2. ˈʃǫb.bjo. ˈknuːl. ˈloj.
ˈnoǫs.luːn. ˈjeg. ˈjel.luːl.
ˈtsweǫ.na. ˈmin. moʧ.ˈmoǫs.
ˈgeǫ. tsel.ˈkols.

South America

1. ˈĩʧ. ˈwe.sa̰m. ˈbut.hḭt.
ˈhḭd.ku. ˈmaw.tḭr. ˈwo.wa̰. ˈpi.ʧã̰s.
ˈaw.ʧa. ˈɾuɾ.wu. ˈha.me. ˈweɾ.

2. ˈutʃ. ˈhɨ.ɾĩ. ˈnaw.ʧoɾ.
ˈɨʧ.ʧe. ˈhɨ. ˈmɨr.ʧu. ˈta̰ɾ.
ˈɾiw. ˈwi.te. ˈkɨh.
ˈʧĩ. ˈɨʧ.ɾed. ˈwu.ʧi.

Europe

1. ˈʂuːf.räɾk. ˈʒə. ˈʎä.
ˈʃɔj. ˈnyf. ˈnə.jib. ˈjɔj.juj.
ˈcuh. ˈtəj. ˈmirɖ̪
ˈɲæx.xiʎ. ˈŋejʂ̪nəɲ. ˈmɔdz.

2. ˈlijtʃ. ˈreˌc. ˈʒä. ˈtsoˌh.eg.
ˈhyv.ol. ˈbɛt.sjojr. ˈtsik.
ˈŋo̞.vəj. ˈʧo̞j. ˈljɛɲ. ˈtsryh.yr.
ˈʈ̪ær.ʒjuːʧ. ˈʧjarp.o. ˈdzɔts.tsy.

MENA

1. ˈħi. ˈʧɛʃ.ʊj. ˈkew.
ˈol. ˈxwu.ɖʒil. ˈmi.raf.
ˈluː.jag. ˈɖʒej. ˈweː.be. ˈwʊl.
ˈseː. ˈfrʊ. ˈhetʃ. ˈwaj.gew.

2. ˈʈ̪es. ˈer.la. ˈħɛz.
ˈqu.brʊ. ˈruːk. ˈɖaɳ̪
ˈɣa. ˈvatʃ.gul. ˈħuːw.kra.
ˈɛl. ˈlɛ.ɣuːw. ˈgwɛ.
ˈwor. ˈpuː.ruː.

Sub-Saharan Africa

1. ˈkpeːɬʧ. ˈbeɬ.ziːɬ. ˈɖʒaɬ. ˈəɬɲ.
ˈzoɬ.ʧuɬgb. ˈkpjeɬ. ˈjiɬgb. ˈneːɬk.ɔɬj.
ˈʧeɬʧ.loɬ. ˈɔɬw. ˈɖʒuɬh.ɲaɬw.
ˈvəɬk. ˈteɬ.jaɬ. ˈiːɬɓ.muɬs. ˈɖʒlaɬ.

2. ˈləɬg.deɬ. ˈɓmuɬ.weɬm. ˈləɬ.ɲaɬf.
ˈviɬw.wəɬ. ˈloɬ. ˈɓeːɬj.
ˈseɬz. ˈfɲɛɬ. ˈʧmiːɬ.vəɬz.
ˈləɬ.tɛɬl. ˈiːɬl. ˈwiɬh.

India

1. ˈkʰju.cçʊl. ˈɖiːk. ˈsʊ.
ˈriːɻɻ.r̃i. ˈʈiː.dar. ˈmʊ.
ˈŋr̃cç. ˈsroː.ro. ˈɖɛg.boːpʰ.
ˈzʊɖ. ˈroːɖ. ˈɻɻ.i̥ŋ.ɻɻo.

2. ˈŋi. ˈto. ʈoːr.ˈrən.
ˈsu.ŋɪ. ˈljʊ.ʊn. ˈpuʈ pʰjɪ.ˈlə.
ˈliːr. ˈcçel.re. ˈloː. ˈnin.zəl.
ˈhɔʈ.jɪr. ɖɛs.ˈɲiːɻɻ. ˈəg.

East Asia

1. lɑtʰ.ˈgar. ˈko. lɛtʰ.ˈtsʰe̞h. re.ˈmjoː.
ˈsɑl.ʧʰe̞. ˈwij.mir. pʰə.ˈtsuːŋ. ɑʃ.ˈsnɯ.
ˈdos. ˈru.pyɲ. ˈʈef.ŋa. ˈrɛ.

2. ˈja. ˈre̞ʞʰ.wər. ˈɲɯr.rɑʈʃ. ˈŋi.rɛts.
zo.ˈgɑ. ˈtʰɑm. ˈpʰuːw.ban.
ˈfɯʃ. ŋe̞.ˈku. ˈnɑ.
ˈnɔʧʰ. ˈlol. ˈʃe̞l.

Southeast Asia

1. ˈŋɔt. ˈwɯl. ˈɲi. ˈlu.lu.
ˈhle.lɔs. ˈhus.mɛ. ˈsiː. ˈtmiː.
ˈlɛw.dle. ˈbo.ɲwɯ. ˈwe.gŋɛ.
ˈmɔ. ˈho.swɔ. ˈwɔw. ˈməh.

2. ˈlɯ.hu. ˈlo.ŋa. ˈwo.
ˈŋɛ.nɔ. ˈiː. ˈpʰiw. ˈgwi.kiː.
ˈslo.hwo. ˈwit. ˈwe.
ˈɔm.wɯɲ. ˈgwɔ.

Australia

1. ˈɭun.ta. ˈɭu.pu. ˈɭaw.wa.
ˈniːŋ. ˈiːw.ji. ˈŋʖwa.ŋʖak. ˈkiɭ.
ˈɭi.liɭ. ˈwiː.ljiː. ˈmu.ɭu. ˈtwi.ƚiːɭ.

2. ˈɭaƚ. ˈŋiː. ˈla.liːɭ.
ˈjiː. ˈŋa.uw. ˈjiːƚ.ƚu. ˈtwi.kiː.
ˈpa. ˈŋi. ˈwa.jiːƚ. ˈɳiw.
ˈʈu.ŋa. ˈwip. ˈta.

Appendix C - Code for Phonology and Sample Generation

All code was executed in an Anaconda Notebook running Python 3.9.7.

```
# Defining two functions that can be used to create macroareas, or add to existing macroareas.
# These functions are meant to be used with the following files:
# WALS "WALSLanguages.csv": https://wals.info/languoid
# PHOIBLE's "phoible.csv": https://github.com/phoible/dev/tree/master/data
# PHOIBLE's "PhoibleLanguages.csv":https://phoible.org/languages

import pandas as pd

def new_macroarea_by_latlong(wals_source, phoible_source, phoible_lang_source, min_latitude, max_latitude,
min_longitude, max_longitude):
    new_macroarea = []
    walsLanguages = pd.read_csv(wals_source)
    for language in walsLanguages.index:
        iso_code = walsLanguages.at[language, 'iso_codes']
        latitude = walsLanguages.at[language, 'latitude']
        longitude = walsLanguages.at[language, 'longitude']
        if latitude > min_latitude and latitude < max_latitude and longitude > min_longitude and longitude <
max_longitude:
            if iso_code not in new_macroarea:
                new_macroarea.append(iso_code)

    phoible = pd.read_csv(phoible_source)
    phoibleLanguages = pd.read_csv(phoible_lang_source)
    for language in walsLanguages.index:
        langID = phoibleLanguages.at[language, 'id']

        latitude = phoibleLanguages.at[language, 'latitude']
        longitude = phoibleLanguages.at[language, 'longitude']
        if latitude > min_latitude and latitude < max_latitude and longitude > min_longitude and longitude <
max_longitude:
            for entry in phoible.index:
                if phoible.at[entry, 'Glottocode'] == langID:
                    iso_code = phoible.at[entry, 'ISO6393']
                    break
                else:
                    continue
```

```
        if iso_code not in new_macroarea:
            new_macroarea.append(iso_code)

    return new_macroarea

def append_macroarea_by_latlong(existing_macroarea, wals_source, phoible_source, phoible_lang_source,
min_latitude, max_latitude, min_longitude, max_longitude):

    walsLanguages = pd.read_csv(wals_source)
    for language in walsLanguages.index:
        iso_code = walsLanguages.at[language, 'iso_codes']
        latitude = walsLanguages.at[language, 'latitude']
        longitude = walsLanguages.at[language, 'longitude']
        if latitude > min_latitude and latitude < max_latitude and longitude > min_longitude and longitude <
max_longitude:
            if iso_code not in existing_macroarea:
                existing_macroarea.append(iso_code)

    phoible = pd.read_csv(phoible_source)
    phoibleLanguages = pd.read_csv(phoible_lang_source)
    for language in walsLanguages.index:
        langID = phoibleLanguages.at[language, 'id']

        latitude = phoibleLanguages.at[language, 'latitude']
        longitude = phoibleLanguages.at[language, 'longitude']
        if latitude > min_latitude and latitude < max_latitude and longitude > min_longitude and longitude <
max_longitude:
            for entry in phoible.index:
                if phoible.at[entry, 'Glottocode'] == langID:
                    iso_code = phoible.at[entry, 'ISO6393']
                    break
                else:
                    continue

            if iso_code not in existing_macroarea:
                existing_macroarea.append(iso_code)
# Defining two functions that can be used to create macroareas, or add to existing macroareas.
# These functions are meant to be used with the following files:
# WALS "WALSLanguages.csv": https://wals.info/languoid
# PHOIBLE's "phoible.csv": https://github.com/phoible/dev/tree/master/data
# PHOIBLE's "PhoibleLanguages.csv":https://phoible.org/languages

import pandas as pd
```

```
def new_macroarea_by_latlong(wals_source, phoible_source, phoible_lang_source, min_latitude, max_latitude,
min_longitude, max_longitude):
    new_macroarea = []
    walsLanguages = pd.read_csv(wals_source)
    for language in walsLanguages.index:
        iso_code = walsLanguages.at[language, 'iso_codes']
        latitude = walsLanguages.at[language, 'latitude']
        longitude = walsLanguages.at[language, 'longitude']
        if latitude > min_latitude and latitude < max_latitude and longitude > min_longitude and longitude <
max_longitude:
            if iso_code not in new_macroarea:
                new_macroarea.append(iso_code)

    phoible = pd.read_csv(phoible_source)
    phoibleLanguages = pd.read_csv(phoible_lang_source)
    for language in walsLanguages.index:
        langID = phoibleLanguages.at[language, 'id']

        latitude = phoibleLanguages.at[language, 'latitude']
        longitude = phoibleLanguages.at[language, 'longitude']
        if latitude > min_latitude and latitude < max_latitude and longitude > min_longitude and longitude <
max_longitude:
            for entry in phoible.index:
                if phoible.at[entry, 'Glottocode'] == langID:
                    iso_code = phoible.at[entry, 'ISO6393']
                    break
                else:
                    continue

            if iso_code not in new_macroarea:
                new_macroarea.append(iso_code)

    return new_macroarea

def append_macroarea_by_latlong(existing_macroarea, wals_source, phoible_source, phoible_lang_source,
min_latitude, max_latitude, min_longitude, max_longitude):

    walsLanguages = pd.read_csv(wals_source)
    for language in walsLanguages.index:
        iso_code = walsLanguages.at[language, 'iso_codes']
        latitude = walsLanguages.at[language, 'latitude']
        longitude = walsLanguages.at[language, 'longitude']
        if latitude > min_latitude and latitude < max_latitude and longitude > min_longitude and longitude <
max_longitude:
```

```
        if iso_code not in existing_macroarea:
            existing_macroarea.append(iso_code)

    phoible = pd.read_csv(phoible_source)
    phoibleLanguages = pd.read_csv(phoible_lang_source)
    for language in walsLanguages.index:
        langID = phoibleLanguages.at[language, 'id']

        latitude = phoibleLanguages.at[language, 'latitude']
        longitude = phoibleLanguages.at[language, 'longitude']
        if latitude > min_latitude and latitude < max_latitude and longitude > min_longitude and longitude <
max_longitude:
            for entry in phoible.index:
                if phoible.at[entry, 'Glottocode'] == langID:
                    iso_code = phoible.at[entry, 'ISO6393']
                    break
                else:
                    continue

            if iso_code not in existing_macroarea:
                existing_macroarea.append(iso_code)

# Defining the Europe macroarea
macroarea_europe = new_macroarea_by_latlong("WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv",
36, 72, -12, 28)
# Iceland
append_macroarea_by_latlong(macroarea_europe, "WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv",
62, 67, -25, -11)
#Faroe Islands
append_macroarea_by_latlong(macroarea_europe, "WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv",
76, 81, 10, 33)
#Eastern Europe and Russia
append_macroarea_by_latlong(macroarea_europe, "WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv",
43, 78, 21, 60)

# NORTH AMERICA
# Continental USA and Canada
macroarea_nAmerica = new_macroarea_by_latlong("WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv",
32.6, 84, -140, -30)
# Greenland
append_macroarea_by_latlong(macroarea_nAmerica, "WALSLanguages.csv", "phoible.csv",
"PhoibleLanguages.csv", 68, 84, -75, -8)
# Alaska
```

```
append_macroarea_by_latlong(macroarea_nAmerica, "WALSLanguages.csv", "phoible.csv",
"PhoibleLanguages.csv", 53, 72, -169, -140)
# Southern US and Florida
append_macroarea_by_latlong(macroarea_nAmerica, "WALSLanguages.csv", "phoible.csv",
"PhoibleLanguages.csv", 25, 35, -100, -79)
# American Southwest
append_macroarea_by_latlong(macroarea_nAmerica, "WALSLanguages.csv", "phoible.csv",
"PhoibleLanguages.csv", 29, 35, -120, -100)

# CENTRAL AMERICA
# Most of Mexico and Central America
macroarea_cAmerica = new_macroarea_by_latlong("WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv",
10, 26, -120, -83)
# Costa Rica and Panama
append_macroarea_by_latlong(macroarea_cAmerica, "WALSLanguages.csv", "phoible.csv",
"PhoibleLanguages.csv", 7, 11, -86, -76)
# Northwest Mexico
append_macroarea_by_latlong(macroarea_cAmerica, "WALSLanguages.csv", "phoible.csv",
"PhoibleLanguages.csv", 26, 29, -104, -98)
append_macroarea_by_latlong(macroarea_cAmerica, "WALSLanguages.csv", "phoible.csv",
"PhoibleLanguages.csv", 22, 32, -120, -104)
#Caribbean
append_macroarea_by_latlong(macroarea_cAmerica, "WALSLanguages.csv", "phoible.csv",
"PhoibleLanguages.csv", 12, 25, -86, -60)

# SOUTH AMERICA
macroarea_sAmerica = new_macroarea_by_latlong("WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv",
-57, 13, -82, -31)

# MIDDLE EAST AND NORTH AFRICA (MENA)
# North Africa
macroarea_mena = new_macroarea_by_latlong("WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv", 19,
37, -20, 39)
# Middle East
append_macroarea_by_latlong(macroarea_mena, "WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv",
22, 41, 27, 63)
# Arabian Peninsula
append_macroarea_by_latlong(macroarea_mena, "WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv",
12.5, 30, 43, 60)
append_macroarea_by_latlong(macroarea_mena, "WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv",
16.5, 30, -40, 41)

#SUBSAHARAN AFRICA
# Most of Mainland Africa
```

```
macroarea_africa = new_macroarea_by_latlong("WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv", -37, 19, -20, 40.5)
# Somalia
append_macroarea_by_latlong(macroarea_africa, "WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv", -6, 15.5, 35, 52)
# Madagascar
append_macroarea_by_latlong(macroarea_africa, "WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv", -28, -10, 42, 53)

# INDIA
# India and Bangladesh
macroarea_india = new_macroarea_by_latlong("WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv", 4, 30, 61, 97)
# Pakistan and Afghanistan
append_macroarea_by_latlong(macroarea_india, "WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv", 25, 37, 61, 80)

# EAST ASIA
# Most of East Asia
macroarea_eastAsia = new_macroarea_by_latlong("WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv", 28, 52, 79, 148)
# Western Xinjiang
append_macroarea_by_latlong(macroarea_eastAsia, "WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv", 34, 42, 73.5, 79)
# South China, Taiwan, Ryukyu
append_macroarea_by_latlong(macroarea_eastAsia, "WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv", 22, 28.5, 98, 134)
# Guangdong (incl. Hainan)
append_macroarea_by_latlong(macroarea_eastAsia, "WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv", 18, 25, 108, 118)

# SOUTHEAST ASIA
# Most of Southeast Asia
macroarea_southeastAsia = new_macroarea_by_latlong("WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv", -10, 18, 92, 110)
# Myanmar
append_macroarea_by_latlong(macroarea_southeastAsia, "WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv", 15, 28.5, 93, 99.5)
# North Vietnam
append_macroarea_by_latlong(macroarea_southeastAsia, "WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv", 16, 23, 101, 107)
# Most of Austronesia
append_macroarea_by_latlong(macroarea_southeastAsia, "WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv", -11, 8, 107, 130)
```

```
# Philippines
append_macroarea_by_latlong(macroarea_southeastAsia, "WALSLanguages.csv", "phoible.csv",
"PhoibleLanguages.csv", 3, 21, 114, 128)

# AUSTRALIA
# Australia
macroarea_australia = new_macroarea_by_latlong("WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv",
-45, -11, 111, 155)
# New Zealand
append_macroarea_by_latlong(macroarea_australia, "WALSLanguages.csv", "phoible.csv", "PhoibleLanguages.csv",
-48, -33, 165, 180)

# Defining our relevant phoneme classes

import pandas as pd

phoible = pd.read_csv("phoible.csv")

#BASIC CATEGORIES
consonants = []
vowels = []
tones = []
#MANNERS
nasals = []
plosives = []
fricatives = []
affricates = []
approximants = []
trills = []
taps = []
clicks = []
lateralApproximants = []
lateralFricatives = []
#PLACES
labials = []
labiodentals = []
alveolars = []
dentals = []
retroflexes = []
postalveolars = []
palatals = []
velars = []
uvulars = []
laryngeals = []
```

```
glottals = []
#CONSONANT VOICING
voicedConsonants = []
voicelessConsonants = []
aspiratedConsonants = []
ejectiveConsonants = []
implosiveConsonants = []
#VOWEL PARAMETERS
voicedVowels = []
voicelessVowels = []
nasalVowels = []
#VOWEL LENGTHS
shortVowels = []
longVowels = []

for entry in phoible.index:

   #BASIC CATEGORIES
   #Consonants
   if phoible.at[entry, "SegmentClass"] == "consonant":
      if phoible.at[entry, "Phoneme"] not in consonants:
         consonants.append(phoible.at[entry, "Phoneme"])
   #Vowels
   if phoible.at[entry, "SegmentClass"] == "vowel":
      if phoible.at[entry, "Phoneme"] not in vowels:
         vowels.append(phoible.at[entry, "Phoneme"])
   #Tones
   if phoible.at[entry, "SegmentClass"] == "tone":
      if phoible.at[entry, "Phoneme"] not in tones:
         tones.append(phoible.at[entry, "Phoneme"])

   # CONSONANT MANNERS OF ARTICULATION
   #Nasal
   if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "nasal"] == "+":
      if phoible.at[entry, "Phoneme"] not in nasals:
         nasals.append(phoible.at[entry, "Phoneme"])
   #Plosive
   if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "continuant"] == "-" and
phoible.at[entry, "lateral"] == "-" and phoible.at[entry, "delayedRelease"] == "-":
      if phoible.at[entry, "Phoneme"] not in plosives:
         plosives.append(phoible.at[entry, "Phoneme"])
   #Fricative
   if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "continuant"] == "+" and
phoible.at[entry, "lateral"] == "-" and phoible.at[entry, "approximant"] == "-":
```

```
    if phoible.at[entry, "Phoneme"] not in fricatives:
      fricatives.append(phoible.at[entry, "Phoneme"])
  #Affricate
  if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "continuant"] == "-" and
phoible.at[entry, "lateral"] == "-" and phoible.at[entry, "delayedRelease"] == "+":
    if phoible.at[entry, "Phoneme"] not in affricates:
      affricates.append(phoible.at[entry, "Phoneme"])
  #Approximant
  if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "approximant"] == "+" and
phoible.at[entry, "trill"] == "-" and phoible.at[entry, "tap"] == "-" and phoible.at[entry, "lateral"] == "-":
    if phoible.at[entry, "Phoneme"] not in approximants:
      approximants.append(phoible.at[entry, "Phoneme"])
  #Trill
  if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "trill"] == "+":
    if phoible.at[entry, "Phoneme"] not in trills:
      trills.append(phoible.at[entry, "Phoneme"])
  #Tap
  if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "tap"] == "+":
    if phoible.at[entry, "Phoneme"] not in taps:
      taps.append(phoible.at[entry, "Phoneme"])
  #Click
  if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "click"] == "+":
    if phoible.at[entry, "Phoneme"] not in clicks:
      clicks.append(phoible.at[entry, "Phoneme"])
  #Lateral Approximant
  if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "lateral"] == "+" and phoible.at[entry,
"approximant"] == "+":
    if phoible.at[entry, "Phoneme"] not in lateralApproximants:
      lateralApproximants.append(phoible.at[entry, "Phoneme"])
  #Lateral Fricative
  if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "lateral"] == "+" and phoible.at[entry,
"approximant"] == "-" and phoible.at[entry, "click"] == "-":
    if phoible.at[entry, "Phoneme"] not in lateralFricatives:
      lateralFricatives.append(phoible.at[entry, "Phoneme"])

  #CONSONANT PLACES OF ARTICULATION
  #Labial
  if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "labial"] == "+":
    if phoible.at[entry, "Phoneme"] not in labials:
      labials.append(phoible.at[entry, "Phoneme"])
  #Labiodental
  if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "labiodental"] == "+":
    if phoible.at[entry, "Phoneme"] not in labiodentals:
      labiodentals.append(phoible.at[entry, "Phoneme"])
```

```
    #Alveolar
    if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "coronal"] == "+" and phoible.at[entry, "anterior"] == "+" and phoible.at[entry, "distributed"] == "-":
        if phoible.at[entry, "Phoneme"] not in alveolars:
            alveolars.append(phoible.at[entry, "Phoneme"])
    #Dental
    if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "coronal"] == "+" and phoible.at[entry, "anterior"] == "+" and phoible.at[entry, "distributed"] == "+":
        if phoible.at[entry, "Phoneme"] not in dentals:
            dentals.append(phoible.at[entry, "Phoneme"])
    #Retroflex
    if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "coronal"] == "+" and phoible.at[entry, "anterior"] == "-" and phoible.at[entry, "dorsal"] == "-" and phoible.at[entry, "distributed"] == "-":
        if phoible.at[entry, "Phoneme"] not in retroflexes:
            retroflexes.append(phoible.at[entry, "Phoneme"])
    #Post-alveolar
    if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "coronal"] == "+" and phoible.at[entry, "distributed"] == "+" and phoible.at[entry, "dorsal"] == "-" and phoible.at[entry, "anterior"] == "-":
        if phoible.at[entry, "Phoneme"] not in postalveolars:
            postalveolars.append(phoible.at[entry, "Phoneme"])
    #Palatal
    if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "dorsal"] == "+" and phoible.at[entry, "front"] == "+":
        if phoible.at[entry, "Phoneme"] not in palatals:
            palatals.append(phoible.at[entry, "Phoneme"])
    #Velar
    if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "dorsal"] == "+" and phoible.at[entry, "high"] == "+" and phoible.at[entry, "front"] == "-":
        if phoible.at[entry, "Phoneme"] not in velars:
            velars.append(phoible.at[entry, "Phoneme"])
    #Uvular
    if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "dorsal"] == "+" and phoible.at[entry, "back"] == "+":
        if phoible.at[entry, "Phoneme"] not in uvulars:
            uvulars.append(phoible.at[entry, "Phoneme"])

    #Laryngeals? (no pharyngeals?)
    if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "epilaryngealSource"] == "+":
        if phoible.at[entry, "Phoneme"] not in laryngeals:
            laryngeals.append(phoible.at[entry, "Phoneme"])
    #Glottals?
    if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "sonorant"] == "-" and phoible.at[entry, "spreadGlottis"] == "+":
        if phoible.at[entry, "Phoneme"] not in glottals:
```

```
    glottals.append(phoible.at[entry, "Phoneme"])

#CONSONANT VOICING
#Voiced
if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "periodicGlottalSource"] == "+":
   if phoible.at[entry, "Phoneme"] not in voicedConsonants:
      voicedConsonants.append(phoible.at[entry, "Phoneme"])
#Voiceless
if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "periodicGlottalSource"] == "-":
   if phoible.at[entry, "Phoneme"] not in voicelessConsonants:
      voicelessConsonants.append(phoible.at[entry, "Phoneme"])
#Aspirated
if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "spreadGlottis"] == "+":
   if phoible.at[entry, "Phoneme"] not in aspiratedConsonants:
      aspiratedConsonants.append(phoible.at[entry, "Phoneme"])
#Ejective
if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "raisedLarynxEjective"] == "+":
   if phoible.at[entry, "Phoneme"] not in ejectiveConsonants:
      ejectiveConsonants.append(phoible.at[entry, "Phoneme"])
#Implosive
if phoible.at[entry, "SegmentClass"] == "consonant" and phoible.at[entry, "loweredLarynxImplosive"] == "+":
   if phoible.at[entry, "Phoneme"] not in implosiveConsonants:
      implosiveConsonants.append(phoible.at[entry, "Phoneme"])


#VOWEL TYPES
#Voiced
if phoible.at[entry, "SegmentClass"] == "vowel" and phoible.at[entry, "periodicGlottalSource"] == "+":
   if phoible.at[entry, "Phoneme"] not in voicedVowels:
      voicedVowels.append(phoible.at[entry, "Phoneme"])
#Voiceless
if phoible.at[entry, "SegmentClass"] == "vowel" and phoible.at[entry, "periodicGlottalSource"] == "-":
   if phoible.at[entry, "Phoneme"] not in voicelessVowels:
      voicelessVowels.append(phoible.at[entry, "Phoneme"])
#Nasal
if phoible.at[entry, "SegmentClass"] == "vowel" and phoible.at[entry, "nasal"] == "+":
   if phoible.at[entry, "Phoneme"] not in nasalVowels:
      nasalVowels.append(phoible.at[entry, "Phoneme"])

#VOWEL LENGTHS
#Short
if phoible.at[entry, "SegmentClass"] == "vowel" and phoible.at[entry, "long"] == "-":
   if phoible.at[entry, "Phoneme"] not in shortVowels:
      shortVowels.append(phoible.at[entry, "Phoneme"])
```

```
    #Long
    if phoible.at[entry, "SegmentClass"] == "vowel" and phoible.at[entry, "long"] == "+":
        if phoible.at[entry, "Phoneme"] not in longVowels:
            longVowels.append(phoible.at[entry, "Phoneme"])

# Let's add all of these lists into one master dictionary. This is necessary for some code later on.

phonologicalCategories = {
    "consonants": consonants,
    "vowels": vowels,
    "tones": tones,

    "nasals": nasals,
    "plosives": plosives,
    "fricatives": fricatives,
    "affricates": affricates,
    "approximants": approximants,
    "trills": trills,
    "taps": taps,
    "clicks": clicks,
    "lateralApproximants": lateralApproximants,
    "lateralFricatives": lateralFricatives,

    "labials": labials,
    "labiodentals": labiodentals,
    "alveolars": alveolars,
    "dentals": dentals,
    "retroflexes": retroflexes,
    "postalveolars": postalveolars,
    "palatals": palatals,
    "velars": velars,
    "uvulars": uvulars,
    "laryngeals": laryngeals,
    "glottals": glottals,

    "voicedConsonants": voicedConsonants,
    "voicelessConsonants": voicelessConsonants,
    "aspiratedConsonants": aspiratedConsonants,
    "ejectiveConsonants": ejectiveConsonants,
    "implosiveConsonants": implosiveConsonants,

    "voicedVowels": voicedVowels,
    "voicelessVowels": voicelessVowels,
    "nasalVowels": nasalVowels,
```

```
   "shortVowels": shortVowels,
   "longVowels": longVowels
}

consonantManner = {"nasals",
   "plosives",
   "fricatives",
   "affricates",
   "approximants",
   "trills",
   "taps",
   "clicks",
   "lateralApproximants",
   "lateralFricatives"}

consonantPlace = {"labials",
   "labiodentals",
   "alveolars",
   "dentals",
   "retroflexes",
   "postalveolars",
   "palatals",
   "velars",
   "uvulars",
   "laryngeals",
   "glottals"}

consonantVoicing = {"voicedConsonants",
   "voicelessConsonants",
   "aspiratedConsonants",
   "ejectiveConsonants",
   "implosiveConsonants"}

vowelType = {"voicedVowels",
   "voicelessVowels",
   "nasalVowels"}

vowelLength = {
   "shortVowels",
   "longVowels"}

# Finally, let's write a function that generates a full phonology for a specified macroarea.
```

```
def phonology_from_macroarea(phoible_source, macroarea, wals_syllables, wals_rhythm):
    phoible = pd.read_csv(phoible_source)
    walsRhythm = pd.read_csv(wals_rhythm)
    walsSyllables = pd.read_csv(wals_syllables)

    # Not every language in WALS is in PHOIBLE, so we will be counting how big the PHOIBLE sample actually is.
    phoibleMacroarea = []
    for entry in phoible.index:
        language = phoible.at[entry, "ISO6393"]
        if language in macroarea:
            if language not in phoibleMacroarea:
                phoibleMacroarea.append(language)

    # We will need to figure out how many languages in this macroarea have at least one phoneme in a given class.
    languagesWithCategory = {}
    for category in phonologicalCategories.keys():
        languagesWithCategory[category] = []

    # We will also be counting all of the phonemes that occur in each category, for each language.
    # We need a blank dictionary for this. It will help us avoid repeats of the same phoneme.
    phonemesInCategory = {}
    for language in phoibleMacroarea:
        phonemesInCategory[language] = {}
        for category in phonologicalCategories.keys():
            phonemesInCategory[language][category] = []

    # Most importantly, we will need to find the frequency of each individual phoneme (sorted by category), across
ALL languages.
    phonemeFrequency = {}
    for category in phonologicalCategories.keys():
        phonemeFrequency[category] = {}
        for phoneme in phonologicalCategories[category]:
            phonemeFrequency[category][phoneme] = 0

    # Now, let's get into it.
    for entry in phoible.index:
        phoneme = phoible.at[entry, "Phoneme"]
        language = phoible.at[entry, "ISO6393"]
        if language in phoibleMacroarea:

            for category in phonologicalCategories.keys():
                if phoneme in phonologicalCategories[category]:

                    if phoneme not in phonemesInCategory[language][category]:
```

```
            phonemesInCategory[language][category].append(phoneme)
            phonemeFrequency[category][phoneme] += 1

        if language not in languagesWithCategory[category]:
            languagesWithCategory[category].append(language)

# Next, we determine which categories are present in at least 60% of languages in this macroarea.
macroareaCategories = []
for category in phonologicalCategories.keys():
    if len(languagesWithCategory[category]) >= (len(phoibleMacroarea) * 0.60):
        macroareaCategories.append(category)

# Out of these, we will find the average number of phonemes in that class (out of languages that have the class)

import math

categorySizes = {}
for category in macroareaCategories:
    totalInCategory = 0
    for language in phonemesInCategory:
        totalInCategory += len(phonemesInCategory[language][category])

    # Phoible contains many phonemes that are attested in only a handful of dialects.
    # Thus, we estimate that the phoneme inventories are inflated by roughly 10%.
    totalInCategory = math.floor(totalInCategory * 0.90)

    averageInCategory = math.floor(totalInCategory / len(languagesWithCategory[category]))
    categorySizes[category] = averageInCategory


# And next, let's generate the most likely phonology that matches these restrictions.

import operator

macroareaPhonology = {}
for category in macroareaCategories:
    sortedCategory = sorted(phonemeFrequency[category].items(), key=operator.itemgetter(1), reverse = True)
    commonCategory = sortedCategory[:categorySizes[category]]
    macroareaPhonology[category] = commonCategory


# Time to clean up. We want to remove any phoneme that is not present across multiple categories.

for category in macroareaPhonology.keys():
```

```
# Consonants should be present in at least one Manner, Place, and Voicing category.

if str(category) in consonantManner:
   for phoneme in macroareaPhonology[category]:

      phonemePlaces = 0
      for place in consonantPlace:
         if place in macroareaPhonology:
            if phoneme in macroareaPhonology[place]:
               phonemePlaces += 1

      if phonemePlaces == 0:
         macroareaPhonology[category].remove(phoneme)
         continue


      phonemeVoicing = 0
      for voicing in consonantVoicing:
         if voicing in macroareaPhonology:
            if phoneme in macroareaPhonology[voicing]:
               phonemeVoicing += 1

      if phonemeVoicing == 0:
         macroareaPhonology[category].remove(phoneme)



if str(category) in consonantPlace:
   for phoneme in macroareaPhonology[category]:

      phonemeManners = 0
      for manner in consonantManner:
         if manner in macroareaPhonology:
            if phoneme in macroareaPhonology[manner]:
               phonemeManners += 1

      if phonemeManners == 0:
         macroareaPhonology[category].remove(phoneme)
         continue


      phonemeVoicing = 0
      for voicing in consonantVoicing:
```

```
            if voicing in macroareaPhonology:
                if phoneme in macroareaPhonology[voicing]:
                    phonemeVoicing += 1

        if phonemeVoicing == 0:
            macroareaPhonology[category].remove(phoneme)



if str(category) in consonantVoicing:
    for phoneme in macroareaPhonology[category]:

        phonemePlaces = 0
        for place in consonantPlace:
            if place in macroareaPhonology:
                if phoneme in macroareaPhonology[place]:
                    phonemePlaces += 1

        if phonemePlaces == 0:
            macroareaPhonology[category].remove(phoneme)
            continue


        phonemeManners = 0
        for manner in consonantManner:
            if manner in macroareaPhonology:
                if phoneme in macroareaPhonology[manner]:
                    phonemeManners += 1

        if phonemeManners == 0:
            macroareaPhonology[category].remove(phoneme)


# Vowels should be present in at least one Type and Length Category.

if str(category) in vowelType:
    for phoneme in macroareaPhonology[category]:

        phonemeLengths = 0
        for length in vowelLength:
            if length in macroareaPhonology:
                if phoneme in macroareaPhonology[length]:
                    phonemeLengths += 1
```

```
            if phonemeLengths == 0:
                macroareaPhonology[category].remove(phoneme)


    if str(category) in vowelLength:
        for phoneme in macroareaPhonology[category]:

            phonemeTypes = 0
            for phonemeType in vowelLength:
                if phonemeType in macroareaPhonology:
                    if phoneme in macroareaPhonology[phonemeType]:
                        phonemeTypes += 1

            if phonemeTypes == 0:
                macroareaPhonology[category].remove(phoneme)



macroareaPhonology['consonants'] = []
for category in macroareaPhonology.keys():
    if category in consonantManner or category in consonantPlace or category in consonantVoicing:
        for phoneme in macroareaPhonology[category]:
            if phoneme not in macroareaPhonology['consonants']:
                macroareaPhonology['consonants'].append(phoneme)


macroareaPhonology['vowels'] = []
for category in macroareaPhonology.keys():
    if category in vowelType or category in vowelLength:
        for phoneme in macroareaPhonology[category]:
            if phoneme not in macroareaPhonology['vowels']:
                macroareaPhonology['vowels'].append(phoneme)

# We also want to get the most common level of syllable complexity and the most common rhythm type.

#First, maximal syllable.
syllableFrequency = {"Simple": 0, "ModeratelyComplex": 0, "Complex": 0}
for entry in walsSyllables.index:
    langID = walsSyllables.at[entry, "id"]
    language = langID.replace("12A-",'')
    if language in macroarea:
        if walsSyllables.at[entry, "domainelement_pk"] == 56:
            syllableFrequency["Complex"] += 1
        elif walsSyllables.at[entry, "domainelement_pk"] == 55:
```

```
            syllableFrequency["ModeratelyComplex"] += 1
        elif walsSyllables.at[entry, "domainelement_pk"] == 54:
            syllableFrequency["Simple"] += 1

    averageSyllable = max(syllableFrequency, key=syllableFrequency.get)
    macroareaPhonology["syllableStructure"] = averageSyllable

    #Second, rhythm type.
    rhythmFrequency = {"Trochaic": 0, "Iambic": 0, "Dual": 0, "None": 0}
    for entry in walsRhythm.index:
        langID = walsRhythm.at[entry, "id"]
        language = langID.replace("17A-",'')
        if language in macroarea:
            if walsRhythm.at[entry, "domainelement_pk"] == 82:
                rhythmFrequency["Trochaic"] += 1
            elif walsRhythm.at[entry, "domainelement_pk"] == 83:
                rhythmFrequency["Iambic"] += 1
            elif walsRhythm.at[entry, "domainelement_pk"] == 84:
                rhythmFrequency["Dual"] += 1
            elif walsRhythm.at[entry, "domainelement_pk"] == 86:
                rhythmFrequency["None"] += 1

    averageRhythm = max(rhythmFrequency, key=rhythmFrequency.get)
    macroareaPhonology["rhythm"] = averageRhythm

    return macroareaPhonology

europePhonology = phonology_from_macroarea("phoible.csv", macroarea_europe, "SyllableStructure.csv",
"RhythmTypes.csv")
nAmericaPhonology = phonology_from_macroarea("phoible.csv", macroarea_nAmerica, "SyllableStructure.csv",
"RhythmTypes.csv")
cAmericaPhonology = phonology_from_macroarea("phoible.csv", macroarea_cAmerica, "SyllableStructure.csv",
"RhythmTypes.csv")
sAmericaPhonology = phonology_from_macroarea("phoible.csv", macroarea_sAmerica, "SyllableStructure.csv",
"RhythmTypes.csv")
menaPhonology = phonology_from_macroarea("phoible.csv", macroarea_mena, "SyllableStructure.csv",
"RhythmTypes.csv")
africaPhonology = phonology_from_macroarea("phoible.csv", macroarea_africa, "SyllableStructure.csv",
"RhythmTypes.csv")
indiaPhonology = phonology_from_macroarea("phoible.csv", macroarea_india, "SyllableStructure.csv",
"RhythmTypes.csv")
australiaPhonology = phonology_from_macroarea("phoible.csv", macroarea_australia, "SyllableStructure.csv",
"RhythmTypes.csv")
```

```
eastAsiaPhonology = phonology_from_macroarea("phoible.csv", macroarea_eastAsia, "SyllableStructure.csv", "RhythmTypes.csv")
southeastAsiaPhonology = phonology_from_macroarea("phoible.csv", macroarea_southeastAsia, "SyllableStructure.csv", "RhythmTypes.csv")

macroareaPhonologies = [europePhonology, nAmericaPhonology, cAmericaPhonology, sAmericaPhonology, menaPhonology, africaPhonology, indiaPhonology, eastAsiaPhonology, southeastAsiaPhonology, australiaPhonology]

# For each phonology, we want to generate a list of phonemes that are emblematic of that region.
# A phoneme is considered emblematic if it only occurs in, at most, one other macroarea.



def identifyEmblematicPhonemes(macroareaPhonologies):

    import re

    emblematicPhonemes = []

    for phonology in macroareaPhonologies:

        phonology['emblematic'] = []

        for category in phonology:
            if category == 'emblematic' or category == 'syllableStructure' or category == 'rhythm':
                continue
            for entry in phonology[category]:

                pattern = "'(.*?)'"
                phoneme = re.search(pattern, str(entry)).group(1)
                phonemeFreq = 0

                for otherPhonology in macroareaPhonologies:
                    if otherPhonology is not phonology:
                        for category in otherPhonology:
                            if category == 'emblematic' or category == 'syllableStructure' or category == 'rhythm':
                                continue

                            for otherEntry in otherPhonology[category]:

                                otherPhoneme = re.search(pattern, str(otherEntry)).group(1)
                                if otherPhoneme == phoneme:
                                    phonemeFreq += 1
```

```
            if phonemeFreq <= 1:
                if entry not in phonology['emblematic']:
                    phonology['emblematic'].append(entry)

        for emblem in phonology['emblematic']:
            occurrence = 0
            for manner in consonantManner:
                if manner in phonology.keys():
                    if emblem in phonology[manner]:
                        occurrence += 1

            if emblem in phonology['vowels']:
                occurrence += 1


            if occurrence == 0:
                phonology['emblematic'].remove(emblem)

identifyEmblematicPhonemes(macroareaPhonologies)

# Before we can generate a string of random syllables, we have to define the sonority hierarchy.

sonority = {'clicks': 0, 'plosives': 0, 'affricates': 0, 'fricatives': 0, 'lateralFricatives': 0,
      'nasals': 3, 'lateralApproximants': 3, 'trills': 3, 'taps': 3, 'approximants': 4}

# This function will generate a string of X syllables that conform to the phonological characteristics of the
macroarea.

import random
import re

def string_from_phonology(phonology, numSyllables):

   syllableList = []
   string = ""

   syllableStructure = phonology['syllableStructure']
   rhythm = phonology['rhythm']

   def generate_simple_syllable():
      newSyllable = ""

      validManners = []
```

```
    for category in list(phonology.keys()):
        if category in consonantManner:
            if len(phonology[category]) > 0:
                validManners.append(str(category))

    validManners.append('None')

    while True:
        manner = random.choice(validManners)
        if manner in consonantManner:

            consonant = str(random.choice(phonology[manner]))
            pattern = "'(.*?)'"
            onset = re.search(pattern, consonant).group(1)

            break
        elif manner == 'None':
            onset = ''
            break
        else:
            continue

    vowel = str(random.choice(phonology['vowels']))
    pattern = "'(.*?)'"
    nucleus = re.search(pattern, vowel).group(1)

    if 'tones' in phonology:
        tone = str(random.choice(phonology['tones']))
        pattern = "'(.*?)'"
        tone = re.search(pattern, tone).group(1)
    else:
        tone = ''

    newSyllable = onset + nucleus + tone + "."

    return newSyllable


def generate_modcomplex_syllable():
    newSyllable = ""

    validManners = []
```

```
for category in list(phonology.keys()):
    if category in consonantManner:
        if len(phonology[category]) > 0:
            validManners.append(str(category))

validManners.append('None')

validSyllables = ['CVC', 'CCV']
chosenSyllable = random.choice(validSyllables)

if chosenSyllable == 'CVC':

    onset = ""
    while True:
        manner = random.choice(validManners)
        if manner in consonantManner:

            consonant = str(random.choice(phonology[manner]))
            pattern = "'(.*?)'"
            onset = re.search(pattern, consonant).group(1)

            break
        elif manner == 'None':
            onset = ""
            break
        else:
            continue


    coda = ""
    while True:
        manner = random.choice(validManners)
        if manner in consonantManner:

            consonant = str(random.choice(phonology[manner]))
            pattern = "'(.*?)'"
            coda = re.search(pattern, consonant).group(1)

            break
        elif manner == 'None':
            coda = ""
            break
        else:
            continue
```

```
    vowel = str(random.choice(phonology['vowels']))
    pattern = "'(.*?)'"
    nucleus = re.search(pattern, vowel).group(1)

    if 'tones' in phonology:
        tone = str(random.choice(phonology['tones']))
        pattern = "'(.*?)'"
        tone = re.search(pattern, tone).group(1)
    else:
        tone = ''

    newSyllable = onset + nucleus + tone + coda + "."



if chosenSyllable == 'CCV':

    firstOnset = ""
    while True:
        firstManner = random.choice(validManners)

        if firstManner in consonantManner:

            consonant = str(random.choice(phonology[firstManner]))
            pattern = "'(.*?)'"
            firstOnset = re.search(pattern, consonant).group(1)

            break
        elif firstManner == 'None':
            firstOnset = ""
            break
        else:
            continue


    secondOnset = ""
    while True:
        secondManner = random.choice(validManners)

        if secondManner in consonantManner:
            if firstManner in consonantManner:
                if sonority[secondManner] <= sonority[firstManner]:
                    secondOnset = ""
```

```
                else:
                    consonant = str(random.choice(phonology[secondManner]))
                    pattern = "'(.*?)'"
                    secondOnset = re.search(pattern, consonant).group(1)

                else:
                    consonant = str(random.choice(phonology[secondManner]))
                    pattern = "'(.*?)'"
                    secondOnset = re.search(pattern, consonant).group(1)

                break
            elif secondManner == 'None':
                secondOnset = ""
                break
            else:
                continue

        vowel = str(random.choice(phonology['vowels']))
        pattern = "'(.*?)'"
        nucleus = re.search(pattern, vowel).group(1)

        if 'tones' in phonology:
            tone = str(random.choice(phonology['tones']))
            pattern = "'(.*?)'"
            tone = re.search(pattern, tone).group(1)
        else:
            tone = ''

        newSyllable = firstOnset + secondOnset + nucleus + tone + "."



    return newSyllable

def generate_complex_syllable():

    newSyllable = ""

    validManners = []

    for category in list(phonology.keys()):
        if category in consonantManner:
            if len(phonology[category]) > 0:
                validManners.append(str(category))
```

```
validManners.append('None')

validSyllables = ['CCCVCC', 'CCVCC', 'CCVC', 'CVCC', 'CVC', 'CCV']
chosenSyllable = random.choice(validSyllables)

if chosenSyllable == 'CCCVCC':
   firstOnset = ""
   while True:
      firstManner = random.choice(validManners)

      if firstManner in consonantManner:

         consonant = str(random.choice(phonology[firstManner]))
         pattern = "'(.*?)'"
         firstOnset = re.search(pattern, consonant).group(1)

         break
      elif firstManner == 'None':
         firstOnset = ""
         break
      else:
         continue


   secondOnset = ""
   while True:
      secondManner = random.choice(validManners)

      if secondManner in consonantManner:
         if firstManner in consonantManner:
            if sonority[secondManner] <= sonority[firstManner]:
               secondOnset = ""
               secondManner = "" # This is to prevent the first and third manner from being identical in rare cases.
            else:
               consonant = str(random.choice(phonology[secondManner]))
               pattern = "'(.*?)'"
               secondOnset = re.search(pattern, consonant).group(1)

         else:
            consonant = str(random.choice(phonology[secondManner]))
            pattern = "'(.*?)'"
            secondOnset = re.search(pattern, consonant).group(1)
```

```
        break
    elif secondManner == 'None':
        secondOnset = ""
        break
    else:
        continue

thirdOnset = ""
while True:
    thirdManner = random.choice(validManners)

    if thirdManner in consonantManner:
        if secondManner in consonantManner:
            if sonority[thirdManner] <= sonority[secondManner]:
                thirdOnset = ""
            else:
                consonant = str(random.choice(phonology[thirdManner]))
                pattern = "'(.*?)'"
                thirdOnset = re.search(pattern, consonant).group(1)

        elif firstManner in consonantManner:
            if sonority[thirdManner] <= sonority[firstManner]:
                thirdOnset = ""
            else:
                consonant = str(random.choice(phonology[thirdManner]))
                pattern = "'(.*?)'"
                thirdOnset = re.search(pattern, consonant).group(1)

        else:
            consonant = str(random.choice(phonology[thirdManner]))
            pattern = "'(.*?)'"
            secondOnset = re.search(pattern, consonant).group(1)

        break

    elif thirdManner == 'None':
        secondOnset = ""
        break
    else:
        continue

vowel = str(random.choice(phonology['vowels']))
pattern = "'(.*?)'"
nucleus = re.search(pattern, vowel).group(1)
```

```
if 'tones' in phonology:
   tone = str(random.choice(phonology['tones']))
   pattern = "'(.*?)'"
   tone = re.search(pattern, tone).group(1)
else:
   tone = ''

ultCoda = ""
while True:
   ultManner = random.choice(validManners)

   if ultManner in consonantManner:

      consonant = str(random.choice(phonology[ultManner]))
      pattern = "'(.*?)'"
      ultCoda = re.search(pattern, consonant).group(1)

      break
   elif ultManner == 'None':
      ultCoda = ""
      break
   else:
      continue


penultCoda = ""
while True:
   penultManner = random.choice(validManners)

   if penultManner in consonantManner:
      if ultManner in consonantManner:
         if sonority[penultManner] <= sonority[ultManner]:
            penultCoda = ""
         else:
            consonant = str(random.choice(phonology[penultManner]))
            pattern = "'(.*?)'"
            penultCoda = re.search(pattern, consonant).group(1)

      else:
         consonant = str(random.choice(phonology[penultManner]))
         pattern = "'(.*?)'"
         penultCoda = re.search(pattern, consonant).group(1)
```

```
            break
        elif penultManner == 'None':
            penultCoda = ""
            break
        else:
            continue

    newSyllable = firstOnset + secondOnset + thirdOnset + nucleus + tone+ penultCoda + ultCoda + "."

if chosenSyllable == 'CCVCC':
    firstOnset = ""
    while True:
        firstManner = random.choice(validManners)

        if firstManner in consonantManner:

            consonant = str(random.choice(phonology[firstManner]))
            pattern = "'(.*?)'"
            firstOnset = re.search(pattern, consonant).group(1)

            break
        elif firstManner == 'None':
            firstOnset = ""
            break
        else:
            continue


    secondOnset = ""
    while True:
        secondManner = random.choice(validManners)

        if secondManner in consonantManner:
            if firstManner in consonantManner:
                if sonority[secondManner] <= sonority[firstManner]:
                    secondOnset = ""
                    secondManner = "" # This is to prevent the first and third manner from being identical in rare cases.
                else:
                    consonant = str(random.choice(phonology[secondManner]))
                    pattern = "'(.*?)'"
                    secondOnset = re.search(pattern, consonant).group(1)

            else:
                consonant = str(random.choice(phonology[secondManner]))
```

```
            pattern = "'(.*?)'"
            secondOnset = re.search(pattern, consonant).group(1)

        break
    elif secondManner == 'None':
        secondOnset = ""
        break
    else:
        continue

vowel = str(random.choice(phonology['vowels']))
pattern = "'(.*?)'"
nucleus = re.search(pattern, vowel).group(1)

if 'tones' in phonology:
    tone = str(random.choice(phonology['tones']))
    pattern = "'(.*?)'"
    tone = re.search(pattern, tone).group(1)
else:
    tone = ''

ultCoda = ""
while True:
    ultManner = random.choice(validManners)

    if ultManner in consonantManner:

        consonant = str(random.choice(phonology[ultManner]))
        pattern = "'(.*?)'"
        ultCoda = re.search(pattern, consonant).group(1)

        break
    elif ultManner == 'None':
        ultCoda = ""
        break
    else:
        continue

penultCoda = ""
while True:
    penultManner = random.choice(validManners)

    if penultManner in consonantManner:
        if ultManner in consonantManner:
```

```
            if sonority[penultManner] <= sonority[ultManner]:
              penultCoda = ""
            else:
              consonant = str(random.choice(phonology[penultManner]))
              pattern = "'(.*?)'"
              penultCoda = re.search(pattern, consonant).group(1)

          else:
            consonant = str(random.choice(phonology[penultManner]))
            pattern = "'(.*?)'"
            penultCoda = re.search(pattern, consonant).group(1)

          break
        elif penultManner == 'None':
          penultCoda = ""
          break
        else:
          continue

      newSyllable = firstOnset + secondOnset + nucleus + tone + penultCoda + ultCoda + "."

  if chosenSyllable == 'CCVC':
      firstOnset = ""
      while True:
        firstManner = random.choice(validManners)

        if firstManner in consonantManner:

          consonant = str(random.choice(phonology[firstManner]))
          pattern = "'(.*?)'"
          firstOnset = re.search(pattern, consonant).group(1)

          break
        elif firstManner == 'None':
          firstOnset = ""
          break
        else:
          continue


      secondOnset = ""
      while True:
        secondManner = random.choice(validManners)
```

```
    if secondManner in consonantManner:
        if firstManner in consonantManner:
            if sonority[secondManner] <= sonority[firstManner]:
                secondOnset = ""
                secondManner = "" # This is to prevent the first and third manner from being identical in rare cases.
            else:
                consonant = str(random.choice(phonology[secondManner]))
                pattern = "'(.*?)'"
                secondOnset = re.search(pattern, consonant).group(1)

        else:
            consonant = str(random.choice(phonology[secondManner]))
            pattern = "'(.*?)'"
            secondOnset = re.search(pattern, consonant).group(1)

        break
    elif secondManner == 'None':
        secondOnset = ""
        break
    else:
        continue

vowel = str(random.choice(phonology['vowels']))
pattern = "'(.*?)'"
nucleus = re.search(pattern, vowel).group(1)

if 'tones' in phonology:
    tone = str(random.choice(phonology['tones']))
    pattern = "'(.*?)'"
    tone = re.search(pattern, tone).group(1)
else:
    tone = ''

ultCoda = ""
while True:
    ultManner = random.choice(validManners)

    if ultManner in consonantManner:

        consonant = str(random.choice(phonology[ultManner]))
        pattern = "'(.*?)'"
        ultCoda = re.search(pattern, consonant).group(1)

        break
```

```
        elif ultManner == 'None':
            ultCoda = ""
            break
        else:
            continue

    newSyllable = firstOnset + secondOnset + nucleus + tone + ultCoda + "."

if chosenSyllable == 'CVCC':
    firstOnset = ""
    while True:
        firstManner = random.choice(validManners)

        if firstManner in consonantManner:

            consonant = str(random.choice(phonology[firstManner]))
            pattern = "'(.*?)'"
            firstOnset = re.search(pattern, consonant).group(1)

            break
        elif firstManner == 'None':
            firstOnset = ""
            break
        else:
            continue

    vowel = str(random.choice(phonology['vowels']))
    pattern = "'(.*?)'"
    nucleus = re.search(pattern, vowel).group(1)

    if 'tones' in phonology:
        tone = str(random.choice(phonology['tones']))
        pattern = "'(.*?)'"
        tone = re.search(pattern, tone).group(1)
    else:
        tone = ''

    ultCoda = ""
    while True:
        ultManner = random.choice(validManners)

        if ultManner in consonantManner:

            consonant = str(random.choice(phonology[ultManner]))
```

```
            pattern = "'(.*?)'"
            ultCoda = re.search(pattern, consonant).group(1)

            break
        elif ultManner == 'None':
            ultCoda = ""
            break
        else:
            continue

    penultCoda = ""
    while True:
        penultManner = random.choice(validManners)

        if penultManner in consonantManner:
            if ultManner in consonantManner:
                if sonority[penultManner] <= sonority[ultManner]:
                    penultCoda = ""
                else:
                    consonant = str(random.choice(phonology[penultManner]))
                    pattern = "'(.*?)'"
                    penultCoda = re.search(pattern, consonant).group(1)

            else:
                consonant = str(random.choice(phonology[penultManner]))
                pattern = "'(.*?)'"
                penultCoda = re.search(pattern, consonant).group(1)

            break
        elif penultManner == 'None':
            penultCoda = ""
            break
        else:
            continue

    newSyllable = firstOnset + nucleus + tone + penultCoda + ultCoda + "."

if chosenSyllable == 'CVC':

    onset = ""
    while True:
        manner = random.choice(validManners)
        if manner in consonantManner:
```

```
            consonant = str(random.choice(phonology[manner]))
            pattern = "'(.*?)'"
            onset = re.search(pattern, consonant).group(1)

            break
        elif manner == 'None':
            onset = ""
            break
        else:
            continue


    coda = ""
    while True:
        manner = random.choice(validManners)
        if manner in consonantManner:

            consonant = str(random.choice(phonology[manner]))
            pattern = "'(.*?)'"
            coda = re.search(pattern, consonant).group(1)

            break
        elif manner == 'None':
            coda = ""
            break
        else:
            continue

    vowel = str(random.choice(phonology['vowels']))
    pattern = "'(.*?)'"
    nucleus = re.search(pattern, vowel).group(1)

    if 'tones' in phonology:
        tone = str(random.choice(phonology['tones']))
        pattern = "'(.*?)'"
        tone = re.search(pattern, tone).group(1)
    else:
        tone = ''

    newSyllable = onset + nucleus + tone + coda + "."



if chosenSyllable == 'CCV':
```

```
firstOnset = ""
while True:
   firstManner = random.choice(validManners)

   if firstManner in consonantManner:

      consonant = str(random.choice(phonology[firstManner]))
      pattern = "'(.*?)'"
      firstOnset = re.search(pattern, consonant).group(1)

      break
   elif firstManner == 'None':
      firstOnset = ""
      break
   else:
      continue


secondOnset = ""
while True:
   secondManner = random.choice(validManners)

   if secondManner in consonantManner:
      if firstManner in consonantManner:
         if sonority[secondManner] <= sonority[firstManner]:
            secondOnset = ""
         else:
            consonant = str(random.choice(phonology[secondManner]))
            pattern = "'(.*?)'"
            secondOnset = re.search(pattern, consonant).group(1)

      else:
         consonant = str(random.choice(phonology[secondManner]))
         pattern = "'(.*?)'"
         secondOnset = re.search(pattern, consonant).group(1)

      break
   elif secondManner == 'None':
      secondOnset = ""
      break
   else:
      continue
```

```
        vowel = str(random.choice(phonology['vowels']))
        pattern = "'(.*?)'"
        nucleus = re.search(pattern, vowel).group(1)

        if 'tones' in phonology:
            tone = str(random.choice(phonology['tones']))
            pattern = "'(.*?)'"
            tone = re.search(pattern, tone).group(1)
        else:
            tone = ''

        newSyllable = firstOnset + secondOnset + nucleus + tone + "."

    return newSyllable

  if phonology['syllableStructure'] == 'Simple':

    while True:
        syllableList = []
        for x in range(numSyllables):
            syllable = generate_simple_syllable()
            syllableList.append(syllable)

        containsAllPhonemes = 'True'
        for category in phonology.keys():
            if category == 'consonants' or category == 'vowels' or category == 'tones': # changed from in
ConsonantManner
                for item in phonology[category]:
                    pattern = "'(.*?)'"
                    t = str(item)
                    phoneme = re.search(pattern, t).group(1)

                    if phoneme not in ' '.join(syllableList):
                        containsAllPhonemes = 'False'

        containsEmblematicTwice = 'True'
        for item in phonology['emblematic']:
            emblematicCount = 0
            pattern = "'(.*?)'"
            t = str(item)
            phoneme = re.search(pattern, t).group(1)
            emblematicCount = ' '.join(syllableList).count(phoneme)
            if emblematicCount < 2:
                containsEmblematicTwice = 'False'
```

```
            break
        else:
            continue

    if containsAllPhonemes == 'False':
        continue

    elif containsEmblematicTwice == 'False':
        continue

    else:
        break

if phonology['syllableStructure'] == 'ModeratelyComplex':

    while True:
        syllableList = []
        for x in range(numSyllables):
            syllable = generate_modcomplex_syllable()
            syllableList.append(syllable)

        containsAllPhonemes = 'True'
        for category in phonology.keys():
            if category in consonantManner or category == 'vowels' or category == 'tones':
                for item in phonology[category]:
                    pattern = "'(.*?)'"
                    t = str(item)
                    phoneme = re.search(pattern, t).group(1)

                    if phoneme not in ' '.join(syllableList):
                        containsAllPhonemes = 'False'

        containsEmblematicTwice = 'True'
        for item in phonology['emblematic']:
            emblematicCount = 0
            pattern = "'(.*?)'"
            t = str(item)
            phoneme = re.search(pattern, t).group(1)
            emblematicCount = ' '.join(syllableList).count(phoneme)
            if emblematicCount < 2:
                containsEmblematicTwice = 'False'
                break
            else:
                continue
```

```
        if containsAllPhonemes == 'False':
            continue

        elif containsEmblematicTwice == 'False':
            continue

        else:
            break

if phonology['syllableStructure'] == 'Complex':

    while True:
        syllableList = []
        for x in range(numSyllables):
            syllable = generate_complex_syllable()
            syllableList.append(syllable)

        containsAllPhonemes = 'True'
        for category in phonology.keys():
            if category in consonantManner or category == 'vowels' or category == 'tones':
                for item in phonology[category]:
                    pattern = "'(.*?)'"
                    t = str(item)
                    phoneme = re.search(pattern, t).group(1)

                    if phoneme not in ' '.join(syllableList):
                        containsAllPhonemes = 'False'

        containsEmblematicTwice = 'True'
        for item in phonology['emblematic']:
            emblematicCount = 0
            pattern = "'(.*?)'"
            t = str(item)
            phoneme = re.search(pattern, t).group(1)
            emblematicCount = ' '.join(syllableList).count(phoneme)
            if emblematicCount < 2:
                containsEmblematicTwice = 'False'
                break
            else:
                continue

        if containsAllPhonemes == 'False':
            continue
```

```
        elif containsEmblematicTwice == 'False':
            continue

        else:
            break



syllableCount = 0
while True:
    word = []
    wordLength = random.randint(1, 2)

    for i in range(wordLength):
        word.append(syllableList[syllableCount])
        syllableCount += 1
        if syllableCount == numSyllables:
            break

    if wordLength == 1:
        letters = list(word[0])
        letters.insert(0, '')
        word[0] = ''.join(letters)



    elif wordLength == 2:
        if phonology['rhythm'] == 'Trochaic':
            letters = list(word[0])
            letters.insert(0, '')
            word[0] = ''.join(letters)

        if phonology['rhythm'] == 'Iambic':
            letters = list(word[1])
            letters.insert(0, '')
            word[1] = ''.join(letters)

        if phonology['rhythm'] == 'Dual' or phonology['rhythm'] == 'None':
            stress = random.randint(0, 1)
            letters = list(word[stress])
            letters.insert(0, '')
            word[stress] = ''.join(letters)
```

```
        word = ''.join(word)


        string += word

        string += ' '

        if syllableCount == numSyllables:
            break
        else:
            continue

    words = string.split()


    wordCount = 0
    while True:
        sentenceLength = random.randint(3, 4)

        wordCount += sentenceLength
        if wordCount > len(words):
            string = ' '.join(words)
            break


        # INSERT CODE FOR RHYTHM HERE

        else:
            words.insert(wordCount, '\n')
            wordCount += 1
            continue

    return string

# Now, just use print(string_from_phonology(phonology, X)) and view the results!
```