Praat scripting I: Basic operations

Dr. Fernanda Barrientos

February 28, 2019

1 Scripting: telling Praat to do the work for you

Let's suppose that you have 500 audio files, each one of them with a 5-sound word. We need to annotate them, measure the vowel formants, and generate a table of values for each vowel. While this might sound like a lot of work, it is not that much if we know how to write a script. A script is a mini-programme that tells Praat something like *Take these files, and do this and that to each one of them.* It sounds too good to be true. But there is a catch: we will need to learn the Praat syntax (which is not that complicated, fortunately). All we need is a bit of patience and attention to detail. In this workshop we will learn the basics of Praat scripting, and will move towards more advanced stuff by the end of it.

2 How does a script work?

In order to make the script do the job, we will need basically three things:

- Define elements: Let's suppose that we want to tell Praat something like *Open all the .wav files from my Downloads folder.* In order to achieve this, we will need to tell Praat the path to your Downloads folder (i.e. something like: `"/home/fernanda/Downloads/MyFolder/"`)
- Give a command: Every programming language has a series of predetermined commands, which will need to be given following a given syntax. For instance, whenever we want Praat to open files, we need to write in the script: `Read from file...` followed by the path to the file.
- Run the script: A script needs to be told that it's time to do the work. In most cases, programmes that allow for scripting have a "Run" button (Praat has one).

Commenting your scripts is always good practice. Commenting means that we can leave a note to ourselves in the script about what that specific bit does (and therefore reuse it, or at least parts of it). But we need to tell the script not to read the comment: in Praat we can to that by starting our comment with a hash symbol (#). Whatever is in that line after that hash symbol will be ignored by Praat when we run the script.

3 Our first script: telling Praat to open several files

Here (https://pandabar.github.io/palabras.zip) you will find a .zip file with 10 .wav files. Each .wav is an audio file containing a word with 5 segments. We will tell Praat to open all these files in one go. We will download this .zip file, extract the files and put the folder with the extracted files on the Desktop.

Now we go to Praat. We go to the Objects window, and then to *Praat → New Praat script*: a blank, editable window will pop up. We write the following bit.

```
# Here we define the directory
sound_directory$ = "/fernanda/Desktop/palabras"

# This will create a list with the names of the files in your folder:
Create Strings as file list... list 'sound_directory$'/*.wav
noFiles = Get number of strings

# And this bit opens the files with the names on the list that we created:
for ifile to noFiles
select Strings list
filename$ = Get string... ifile
Read from file... 'sound_directory$'/'filename$'
endfor
```

The first line defines the directory (i.e. folder) where the files are and that Praat will need to look into. We replace `/fernanda/Desktop/palabras` (my folder) with the path to your own folder. If you don't know it, you can right-click on the folder and then click on Properties (Windows/Linux) or Get Info (Mac).

The second bit creates a list with the names of the files in that folder. We have also created the `noFiles` variable, which is the number of .wav files that your folder has, i.e. 10 (we have told Praat to create this variable by counting the number of file names on the list).

And finally, we have created what in programming language is called a *for loop*: a command where the programme performs the same action over several elements. Hence, what these lines mean is: *Hey Praat: look at the list with the file names. For each name, find the file in the folder that you already know and open it; repeat this action 10 times.*

Now we go to the script menu and save it as a file by going to the script menu: *File* → *Save as...* We make sure that the file extension is .praat. Then hit *Run* → *Run*. This will execute the script that we have created.

4 Automatic TextGrid generation

4.1 But first: inspecting the files

We can easily create a Textgrid for each file with a script. We will have to annotate the each sound file ourselves, but it will still be quicker than the manual option. We know that each file is a word with 5 segments; hence we will create a TextGrid with 5 intervals for each file. However, we need to know how short is the shortest file so that all the boundaries will be within at least that duration.

We will tell Praat the following:

- Create a table where I can see the total duration of each file.
- For each .wav that you opened from my folder, create a TextGrid with two tiers; the first one should have 4 boundaries, spaced evenly every x ms.

For the first operation we will need to create a table first. We will insert in our existent script, RIGHT AFTER the line that says `noFiles = Get number of strings`, the following bit:

```
# This creates a table called "data", with 0 rows (Praat appends them later)
# and two columns.
# One column has the file name, and the other one has the total duration
table_ID = Create Table with column names... data 0 File Duration
```

Since the operation is made over the same files, we can reuse the for loop that we created above. Then, RIGHT BEFORE the line that says `endfor`, we insert this bit:

```
# This fills in the table with rows with file names and the total file duration
end = Get total duration
select table_ID
Append row
Set string value... ifile File 'filename$'
Set numeric value... ifile Duration end
```

Now we inspect the table that appeared in the Objects window. Which is the shortest file? It seems that it is the `nandu.wav` file, with a duration of 0.425 seconds.

4.2 Now for real: the TextGrids

We know that the four boundaries that each TextGrid will have must be within the total duration of the shortest file. It is reasonable then to set four boundaries every 0.1 seconds (we can adjust these boundaries later on, as we annotate the files). So now we will tell Praat: *For each file, create a TextGrid with a tier called* `phone`*. Then place a boundary at 0.1, then a second one at 0.2, then a third one at 0.3, and a fourth one at 0.4.* Hence, RIGHT AFTER where it says `Read from file...'sound_directory$'/'filename$'` we insert the following bit:

```
# This creates a TextGrid with 1 tier and 4 separations for each Sound object.
# Each boundary will be 0.1 s apart from each other.
To TextGrid: "phone", ""
Insert boundary: 1, 0.1
Insert boundary: 1, 0.2
Insert boundary: 1, 0.3
Insert boundary: 1, 0.4
```

This will create one TextGrid for each sound file: they will appear in the Objects window. Now we annotate manually and adjust the boundaries. After we are done with this, we can save the TextGrids to our computer. For this we create the following **new script:**

```
# This script saves all your newly annotated TextGrids in your folder.
# First we select all and tell Praat:
# Every TextGrid in this selection will have a number from 1 to 10
select all
n = numberOfSelected ("TextGrid")
for i to n
tg[i] = selected ("TextGrid", i)
endfor
# Now we tell Praat: take a file and save it as a file with its name
for i to n
selectObject: tg[i]
tg$ = selected$("TextGrid")
Save as text file... 'tg$'.TextGrid
endfor
```

...And then we run this little script with *Run → Run.*

We could also paste this script at the end of the other one, but we will have to be very careful when we run it: instead of using the *Run → Run* option, we will have to select this part with the mouse and then use the option *Run → Run selection* instead. I do not recommend this, because scripting is tiresome and if we are not absolutely aware of what we're doing we will forget about it and will end up saving blank TextGrids, thus losing all the work that was already done (own experience).

5 Obtaining data from a specific file

Now that we have annotated all the files, we would like to know: What is the F1 of the first vowel in `vivir.wav`? Here we will need to extract the sounds from the word in question and the formants as well.

5.1 Extracting sounds automatically

You probably know how to do this manually, but again, if you want to extract one o several sounds per file then it will take a while. We will try with a script instead.

In order to do so, we will need to have our sounds annotated, which we already did in the previous section. Now we create yet another script:

```
# We define our directory
source_directory$ = "/fernanda/Desktop/palabras"

# This creates a table called "data2", with 0 rows and 5 columns.
table_ID2= Create Table with column names... data2 0 Vowel Duration F1 F2

# This bit creates a list with the names of the .wav files in my folder.
wav_files = Create Strings as file list... wavlist 'source_directory$'/*.wav
nowav_files = Get number of strings

# And this creates a list with the names of the .TextGrid files in your folder.
tg_files = Create Strings as file list... tglist 'source_directory$'/*.TextGrid
notg_files = Get number of strings

# This bit opens the .wav files
for ifile to nowav_files
select Strings wavlist
wavfilename$ = Get string... ifile
wav_file= Read from file... 'source_directory$'/'wavfilename$'
palabra$= replace$ (wavfilename$, ".wav", "", 0)

#And this opens the .TextGrid files
select Strings tglist
tgfilename$ = Get string... ifile
tg_file= Read from file... 'source_directory$'/'tgfilename$'
endfor
```

Here we have defined our directory (where the TextGrids and the sound files are), and told Praat: *Create a list with the names of my .wav files, now another list with my .TextGrid files, and open the files with those names.* After running this bit we can look into the Objects window:

we will find the .wav files with their corresponding TextGrids. The first lines created a Table object with 0 rows and 4 columns, which we will fill in with the vowel name, the duration of each vowel, and its corresponding F1 and F2.

Now we will extract the vowels of each word. We paste the following code RIGHT BEFORE the line `endfor`:

```
# This extracts the vowels from each audio file
# and will remove the original audio files from the Objects window.
select wav_file
plus tg_file
Extract intervals where... 1 no "matches (regex)" a|e|i|o|u
select wav_file
plus tg_file
Remove
```

With these lines we have told Praat: *Take a sound, take its TextGrid, and extract the intervals that have the vowel a, e, i, o, u in them.* Then we have also told Praat to remove the original sound files from the Objects window (don't worry -they're still saved in your computer!) and leave only the extracted vowels. This extracting operation will create some Sound objects called e.g.: `Sound pasto_a_e_i_o_u_1`. The final number refers to the vowel in the word: the previous file is the vowel *a* (i.e. the first one) extracted from the word *pasto*. We will rename these sounds to something simpler in the next step.

5.2 Extracting formants

Now that we have extracted the vowels from the words, we can also extract their formants and calculate their respective values automatically. We will create a new for loop, but this time we will tell Praat: *Of all the stuff in the Objects window, take only the Sound files, and assign each one a number.* We paste at the very end of our script the following bit:

```
# This selects all the Sound elements
select all
minusObject: "Table data2", "Strings wavlist", "Strings tglist"
n= numberOfSelected ("Sound")
for i from 1 to n
sound[i] = selected ("Sound", i)
endfor
```

Now we need to: (a) change the name to the files, and (b) know the duration of each one of the vowels. We will also define an element that we will call `mid`: the midpoint of each vowel. Praat will simply take the total duration and divide it by 2.

```
# Change the names of the extracted audio files & calculate their duration
for i from 1 to n
selectObject: sound[i]
sound$=selected$("Sound")
name$=replace$ (sound$, "a_e_i_o_u_", "", 0)
Rename... 'name$'
end= Get total duration
mid= end/2
endfor
```

Now that every sound file is selected, we will tell Praat: Extract the F1 and F2 of each file at the mid point, considering 5 formants within a maximum range of 5,500 Hz[1]. We paste this

[1]If you have a male voice, remember to change this number to 5,000!

RIGHT BEFORE the line `endfor`.

```
# This part calculates F1 and F2
To Formant (burg)... 0.001 5 5500 0.015 50
f1=Get value at time... 1 mid Hertz Linear
f2=Get value at time... 2 mid Hertz Linear
```

But now we want to register all these values. If we recall correctly, in 5.1. we created a table called data2, with 0 rows and the columns Vowel, Duration, F1, F2. Now it's time to add the values of each vowel in the table. We tell Praat: Take the values of each vowel (duration, F1, and F2) and paste them in the table as you go through each file. We paste this code RIGHT BEFORE the `endfor` line:

```
# This enters the values in the table data2
select table_ID2
Append row
Set string value... i Vocal 'name$'
Set numeric value... i Duracion end
Set numeric value... i F1 f1
Set numeric value... i F2 f2
endfor
```

Finally, another good practice regarding Praat scripts is that our script removes all the elements that have been created and leaves the Objects window clear. We do that but keep the table, which we can also save in our folder. We paste this AFTER the `endfor` line:

```
# This bit clears the Objects window and saves the table data2
select all
minusObject: "Table data2"
Remove
SelectObject: "Table data2"
Save as comma-separated file... /formant_table.csv
```

And now most of the job is done: open your folder and you will find a file called "formant_table.csv" with the duration and formant values for each vowel.

6 Doing more stuff with Praat

One of the beauties of scripting is that you can modify scripts that you have already made, so that you don't need to start from scratch every time. Let's suppose that now we want to, say, measure the VOT of the stops: in that case, will have to (a) create an additional tier for your TextGrids, (b) extract the VOTs, and (c) get their total duration:

(a) To create a TextGrid with more tiers, simply go to section 4.2. The script has a line that says:

```
To TextGrid: "phone", ""
```

If you want to create one more tier called VOT, then your line should now read:

```
To TextGrid: "phone VOT", ""
```

Which means: I want two interval tiers, the first one called phone and the second one called VOT. After this, you will have to annotate this tier properly (i.e. giving them intervals that mark the beginning and end of each VOT.)

(b) To extract the VOTs, You may run the last bit in section 5.1. Assuming that your VOT tier in the TextGrid is the second one, and that you marked each VOT simply with the word vot, then the line that says:

```
Extract intervals where... 1 no "matches (regex)" a|e|i|o|u
```

will have to be replaced with following:

```
Extract intervals where... 2 no "is equal to" vot
```

(c) And in order to get the total duration, you may simply reuse and modify the second bit in 5.2.:

```
# Change the names of the extracted audio files & calculate duration
for i from 1 to n
selectObject: sound[i]
end= Get total duration
endfor
```

Here we have modified the part insofar as we don't want to rename the files and that we don't need to know the midpoint of the VOT.

If what you want to do is to measure the overall intensity of each sound (this may be useful if you want to look into the spirantised approximants), then we can modify our script lines again: First we will have to tell Praat that we want to extract the approximants, and not the vowels. How? We look in the code the following line:

```
   Extract intervals where...
   1 no "matches (regex)" a|e|i|o|u
```

Which tells Praat *Extract the intervals in my Textgrid with the annotations a, e, i, o, or u.* We then replace the `a|e|i|o|u` part, with `gf|bf`, which are the labels that we used to annotate the approximants [ɣ] and [β] in the files `trago.wav` and `vivir.wav`. The line will then look like this:

```
# This line will extract approximants instead of vowels
Extract intervals where... 1 no "matches (regex)" gf|bf
```

When we run the script (not yet!) with *Run* → *Run*, we will get an error message. But this is not a problem: it simply means that the word that Praat is currently looking into does not have any interval with such labels. We just hit OK and carry on. This message will appear as many times as we have words without any of these approximants (8 times in this case).

And now, we finally measure the intensity. We replace the line `mid=end/2` (insofar as we don't need to know the midpoint of the approximants) with the following line:

```
# This line calculates the intensity of each sound in dB
int= Get intensity (dB)
```

And in the line where we created the Table with the data we can add a column for the intensity, which we will call `dB`. The line would then look like this:

```
table_ID2= Create Table with column names... data2 0 Segment Duration F1 F2 dB
```

And in the part of the script where we added the calculated values to the table we now add the dB values. Then, right BELOW the line:

```
Set numeric value... i F2 f2
```

We add the following line:

```
# This line adds the intensity of each sound in dB
Set numeric value... i dB int
```

The we run the script, and we will get a table with the data corresponding to the approximants: segment, duration, F1, F2, and dB.

7 Cheating

You can also "cheat" with Praat. This means that whenever we don't know how to proceed with the syntax or what the command is, we can do it manually with a file, and then go to the script menu: *Edit* → *Paste history.* This will paste on the script all the bit that you did manually.

8 Praat syntax

What is "Praat syntax"? In all fairness, it is not a specific syntax per se. When we write a script, what we are basically doing is to replicate what you see in a given dialogue window when you do things manually. Let's create a TextGrid for the Object `Sound cancha`. Then select both elements (Sound and TextGrid) with the mouse and go to *Extract* → *Extract intervals where....* Look at the dialogue window and compare with this line code from Section 6(b):

```
        Extract intervals where... 2 no "is equal to" vot
```

So, the main idea is that Praat scripts are sequences of button-pressing commands. This makes things easier to a very large extent: it means that, if we have 50% of the job done if we know the dialogue boxes that Praat shows to us whenever we do something manually. This is also why you can "cheat" by pasting the history on a given script.

The elements that can be considered as proper syntax so far are the loops, the definitions (e.g. when we have defined our folder; note that we have used a $ symbol for the string variables); and elements for regular expressions, such as in section 5.1.: `a|e|i|o|u` , where the vertical bar stands for "or". All the symbols and correct syntax for this can be found in the Praat manual, along with some examples.