C Praat scripting

A script is a text that contains Praat menu and action commands. When you run the script, all actions and commands will be executed. A script can be useful in various circumstances.

- To automate repetitive actions. You have to do the same series of analyses on a large corpus and you don't want to sit for months at the computer clicking away to do your analyses on thousands of files. Instead, you write a script that performs all necessary steps, for example, reading a sound file from disk, performing the analysis and saving the results. Test the script thoroughly on a small number of files and then order Praat to run the script on all the files in the corpus. You sit back when all the analyses are carried out automatically.

- To fixate a sequence of actions. You have a series of actions on a selected Sound that have a prescribed order. You may script these actions and define a new button in the dynamic menu and every time you click that button, the actions in the associated script are carried out in the right order.

- To log actions. If you want to repeat what you have done, the script serves as a guide.

- To communicate to other people what you have done and how they may achieve the same results. In this book many examples are accompanied by a script.

- To make drawings in the picture window. Nearly all drawings in this book were produced with a script.

We will start by showing you how simple it is to add functionality to Praat once you know how to script. To start scripting you do not have to learn completely new commands to address the functionality of Praat. Simply copy the text that is on the command button. For example if you have created a Sound and want a command in the script to play this Sound, a single script line with only the text "Play" suffices.

C.1 Mistakes to avoid in scripting

- Praat commands in scripts have to be spelled exactly right. Text on menu options and buttons are Praat commands and as you can see, they always starts with an uppercase letter. For example, if you want to play a sound from a script and type "play" instead of "Play" you will receive the message "Unknown command: play" and the script will stop running.

- Sometimes during copy-paste actions or other edit actions, accidentally an extra white space creeps in. For example if you write “Play ”, instead of “Play”, you receive a message “Command “Play” not available for current selection”. This error is often made and hard to detect because you cannot see the extra white space.
- A command exists but is not valid for the selected object. For example, in your script you wanted to “Play” a Sound but accidentally selected a Spectrum object. In this case Praat will show the error message “Command “Play” not available for current selection”.

C.2 Defining a new button in Praat

If you don’t have any sounds in the Object list yet, we first create one. Use the New>Sound>Create Sound from formula... command,[1] click OK and a newly created Sound appears in the list. Now suppose we want the sound to be played twice in a row instead of once. This can be easily done: select the sound and click the “Play” button twice. If we have more sounds which we want to have played twice, we select the next sound, click twice on “Play”, et cetera. In order to avoid rsi, write a script that plays the selected Sound twice. Then bind this script to a newly created button in the dynamic menu. This is a toy problem, but once you know how to script it, you have mastered the basics and can go on to do more complex things.

Start by opening the ScriptEditor. Click on the “New Praat script” command that you find in the leftmost menu item labeled “Praat”. A ScriptEditor window pops up and you type on two consecutive lines the same word “Play”. Figure C.1 gives an impression of the ScriptEditor.

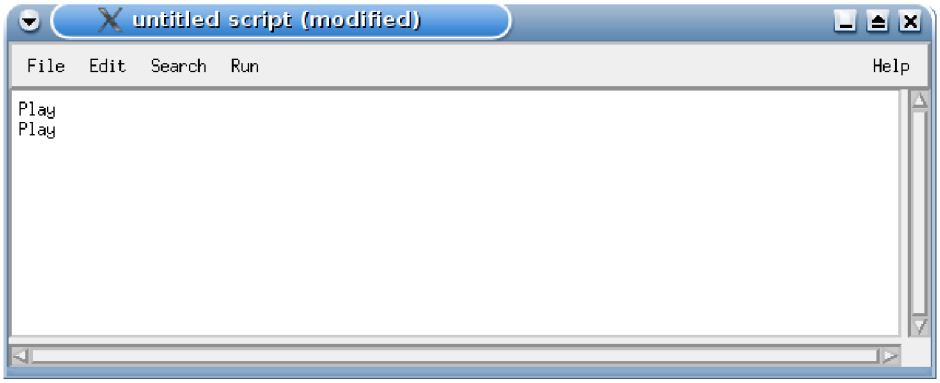

Figure C.1: The ScriptEditor.

Because the script has not been saved yet and we have added two lines, the window bar is titled “untitled script (modified)”. We save the script as a file on the computer disk, with the name `playTwice.praat`, by choosing the “Save as...” command from the File menu which is displayed in figure C.2. Now the title will reflect this name. Select a Sound from the Object

[1] This is a shorthand notation for the actions to find the `Create Sound from formula...` command: click on the New menu, this opens a new list of possible commands. Click on the Sound button in this list. This opens a new list from which you can choose the given command. This notation is only necessary so *you* can find the command, Praat knows where to find its commands.

window. Then choose the “Run” command from the Run menu in the ScriptEditor. You will hear the sound played twice.

Besides these “Play” commands, we can use any other Praat command. In fact, all actions in Praat, i.e. all buttons and all forms, can be addressed in a script, and more. However complex the task you want the script to perform, the same three basic steps are always made. One: open the ScriptEditor. Two: type or copy some commands in the editor. Three: run the script.

We have the script and now we want a new button with the text “Play twice” just underneath the button labeled “Play”. If we click on “Play twice” the script is executed and the selected sound is played twice in a row. From figure C.2 it is clear that the ScriptEditor’s File menu has more entries than “Save as...”. We now choose the “Add to dynamic menu...” command and a

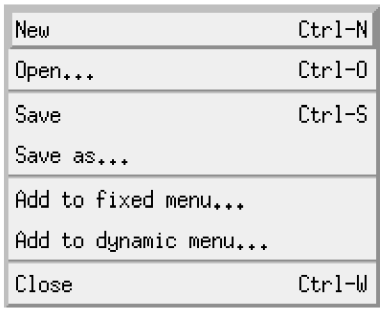

Figure C.2: The ScriptEditor’s File menu.

form like the one displayed in figure C.3 pops up. In the figure two fields have already been modified. One: “Command” originally showed `Do it...` and we changed it into `Play twice`. Two: “After command”, we have filled out `Play`. The last field “Script file” contains the complete file name of your script. File naming is different on Linux, Macintosh and Windows. For example, on Linux, the maps in the path are separated by ’/’ symbols. On Windows maps are separated by the ‘\’ symbol.

After you have clicked the OK button, the dynamic menu changes and a new button with the “Play twice” text appears below the “Play” button. Figure C.4 shows what the upper part of the Sound’s dynamic menu looks like with the new button added.

The newly defined button has the functionality of the script from the ScriptEditor that you associated with it. You may close the ScriptEditor.

C.3 Removing a defined button

The newly defined button works but now we want to remove it. To do this we open the ButtonsEditor that you can reach via the path Praat>Preferences>Buttons.... This editor allows you to determine which buttons will show up and which won’t. It creates the possibility to hide buttons from view that you don’t want to see or never want to use and also to remove buttons that you have added yourself. In this case you want to remove the“ Play twice” button. Because of the thousands of buttons in Praat, the ButtonsEditor will never show all buttons

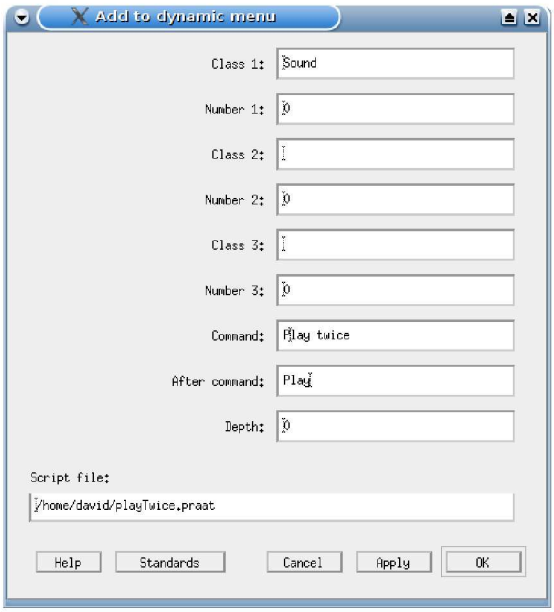

Figure C.3: The ScriptEditor's "Add to dynamic menu..." form.

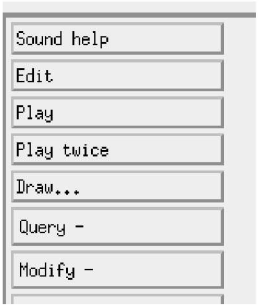

Figure C.4: Part of the dynamic menu for a Sound after a new button has been defined.

at once. Buttons are divided into the categories that are shown in the editor just above the scrollable part. The dynamic menu buttons are divided into two groups of Actions. One for objects that start with a character in the range A-M and one for N-Z. Because the script works on a selected Sound and Sound starts with an "S", we choose the option "Actions N-Z", like the following figure shows. Lots of lines are shown and we scroll until we see the line that starts

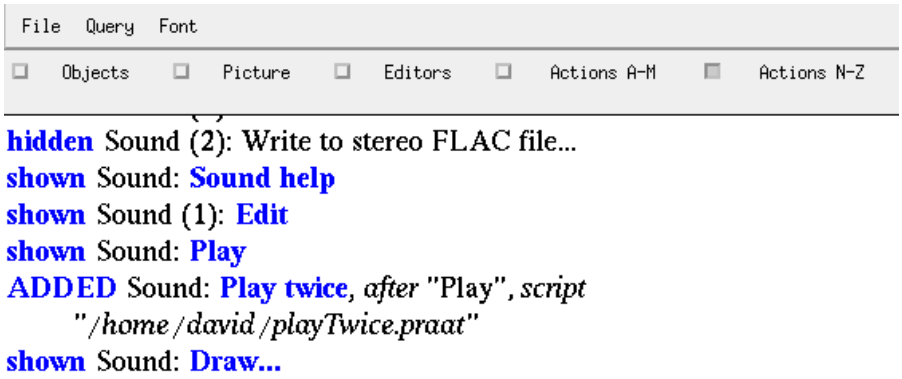

Figure C.5: Part of the ButtonsEditor after first choosing "Actions N-Z" followed by scrolling to the actions for a Sound.

with "ADDED". If we click on "ADDED" the text in the ButtonsEditor immediately changes to "REMOVED" and the "Play twice" button disappears from the dynamic menu. Clicking that same line again, will change the text back to "ADDED" and will make the button reappear.

C.4 Create and play a tone

The previous "Play twice" example was easy. We now create a substantially more powerful script that

- pops up a form to choose a frequency,
- creates a tone[2] of 0.2 seconds with this frequency,
- plays the tone,
- removes the tone.

We will start with a crude version of the script and in a number of small steps improve upon this script until we reach a final version in section C.4.5. In the meantime we introduce the use of variables and how to create simple forms.

The script version we start with will be derived from the commands and actions we perform by pointing and clicking the mouse. Start the New>Sound>Create Sound from formula... command. It will show a form as in figure C.6. Now click OK and a new Sound appears in the list of objects named "sineWithNoise". It is a mono Sound with a duration of 1 second

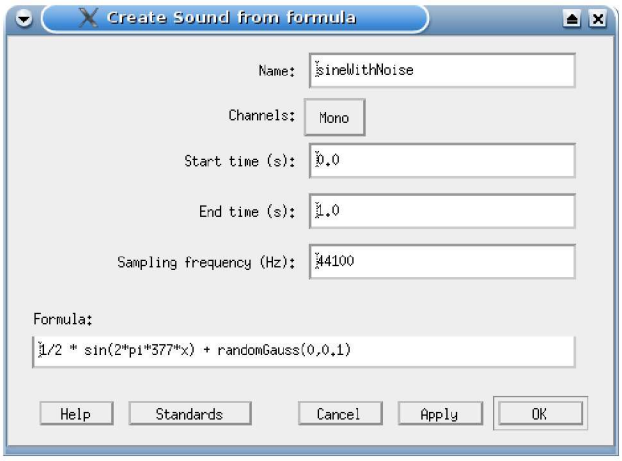

Figure C.6: The Create Sound from formula... form.

and a sampling frequency of 44100 Hz. We listen to this sound by clicking on the "Play" button. It sounds like a pure tone with some noise. We remove the Sound by clicking on the red "Remove" button at the bottom of the Object window.

We start up the ScriptEditor and immediately choose from its Edit menu "Paste history". Now all the commands that we executed in the current Praat session, show in the ScriptEditor. From the moment we start up Praat, the program maintains a history of all the commands and actions we perform. In the ScriptEditor we have access to this history. In the Edit menu of the ScriptEditor we can see two commands that address the history. With one command we can paste the complete history in our script. With the other command we clear the history and start a fresh recording. This history mechanism comes in handy if we want to try out a succession of commands. In that case we start up the ScripEditor, clear the history, perform a number of actions, paste the actions in the ScriptEditot and start editing.

In the script we are working on now, we remove all lines except for the lines that show in the following script.

```
Create Sound from formula... sineWithNoise Mono 0 1 44100
  ... 1/2 * sin(2*pi*377*x) + randomGauss(0,0.1)
Play
Remove
```

If we run this script it repeats the three actions we have just carried out.

Beware: the first two lines in this script were actually one long script line. We have modified this line to fit on the paper. This is something we are always allowed to do with long script lines that don't fit in the normal width of the editor or the paper. There are two things to keep in mind when splitting lines. One: we only split on positions where adding extra white space wouldn't matter. Two: we have to start the continuation part with three consecutive dots (...). These three dots signal that this line is a continuation of the previous line.[3] We may use as many continuation lines as we wish and white space before the three dots is allowed.

[2] A tone is a sound of one frequency.

[3] All commands in Praat that put a form on the screen also end with three dots.

Note that the fields shown vertically in the form, are shown horizontally in the script. We must always maintain this correspondence between the position of a field in a form and the position of the field in a line of text in a script.

We will edit these lines until they do what we want. The first thing is to get rid of the noise. Delete the "+ randomGauss(0,0.1)" text in the "Formula" field. Next we change the "End time" to 0.2. Finally we change the "Name" fields text to 'tone'. The script is now:

```
Create Sound from formula... tone Mono 0 0.2 44100
  ... 1/2 * sin(2*pi*377*x)
Play
Remove
```

When you run this script, the duration of the sound is shorter, it sounds like a pure tone and you won't hear the noise anymore.

The script plays a tone but it is the same tone any time we run the script. We want to vary the tone's frequency. We can do that by typing another number instead of the '377' in the sine function in the Formula field of the first command. Lets say we change '377' into '1000'. If we run the script again, you will hear a higher tone, one of 1000 Hz. By using this script we have actually saved some time. Instead of the three actions: creating a tone, playing the tone and removing the tone, we only have one action now: running the script.

C.4.1 Improvement 1

To change the frequency of the tone we had to edit the number '377' in the formula. A typing error is difficult to locate. Praat, of course, generates an error message with detailed information in which line and approximately where it could not continue the script, but nevertheless, we have to carefully check the formula. It would be nice if we didn't have to change *at all* the most complicated line *in the script* whenever we want another frequency. We can achieve this by introducing *a variable* for the frequency as the following script shows.

```
frequency=1000
Create Sound from formula... tone Mono 0 0.2 44100
  ... 1/2 * sin(2*pi*frequency*x)
Play
Remove
```

The first line introduces a variable with the *name* "frequency" and assigns the *value* "1000" to it.[4] The formula will now use this assigned value, i.e. when the "Create Sound form formula..." evaluates the formula, the value of the variable will be substituted. If you run the modified script, the results will be exactly as they were. However, if you now want to change the frequency, it can be done more easily.

C.4.2 Improvement 2

Now we like to skip editing the script each time we want a tone with a different frequency. We like the script to raise a form in which we can type the desired frequency. The following script improves on what we had.

[4] Variables in a Praat script never start with an uppercase character, commands start with an uppercase character.

```
form Play tone
positive frequency
endform
Create Sound from formula... tone Mono 0 0.2 44100
  ... 1/2*sin(2*pi*frequency*x)
Play
Remove
```

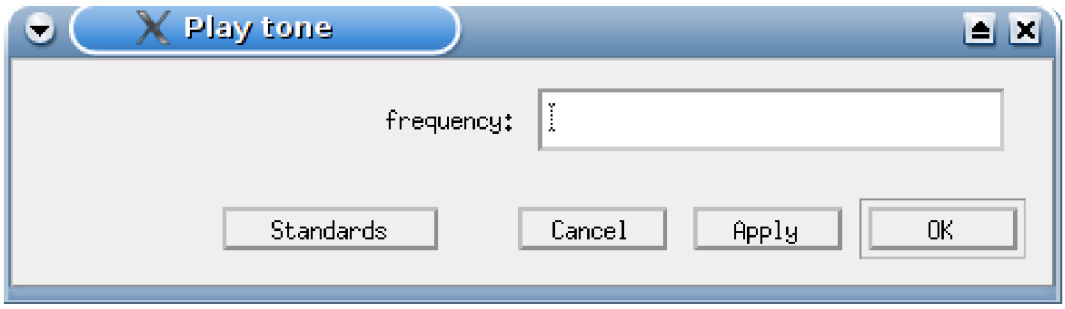

Figure C.7: The first form.

When run, the script raises the form displayed in figure C.7. This form is defined in the first three lines of the script. The first line defines the title for the form, i.e. the text "Play tone" at the top. You can choose your own text, you can even have no text at all.[5] The text of the window should describe or summarize the actions of the script in a compact way. The second line in the script defines a numeric field named "frequency" that allows only *positive* numbers. If you type a number that is less than or equal to zero, a message is generated that will inform you. At the same time a variable with the name "frequency" is created. This frequency *variable* will receive the value of the frequency *field* once OK is clicked. In this way the script and the form communicate: the field name in the form corresponds to a variable with the same name in the script.[6] The endform closes the definition of the form.

Note that all these form definitions *start* with a *lowercase* character. All Praat commands and actions start with an *uppercase* character.

C.4.3 Improvement 3

An annoyance of the previous script is that when the form pops up you have no idea what you should type in the 'frequency' field. If you click OK without typing anything, an error message pops up. It would be nice when the script supplied a default value that guarantees that the scripts runs if the user just clicks OK. Error messages should pop up only if you do something wrong, there is nothing wrong by just accepting defaults. The following script preloads a default value in the frequency field. This number happens to be "377.0".[7]

[5] In the latter case there has to be at least one white space after the 'form' text in the script.

[6] Use the underscore '_' to create white space. For example, the field name 'frequency_value' with associated variable 'frequency_value' shows as 'frequency value' in the form.

[7] To inform the user that real numbers are allowed, it is better to preload with a number that makes this explicit. We therefore used "377.0" instead of "377".

```
form Play tone
positive frequency 377.0
endform
Create Sound from formula... tone Mono 0 0.2 44100
  ... 1/2*sin(2*pi*frequency*x)
Play
Remove
```

The form that pops up is like the form in figure C.7 but now shows the number 377.0 in the frequency field.

C.4.4 Improvement 4

The next improvement is only cosmetic, but nevertheless important. We want to see "Frequency" as the title of the field instead of "frequency", another Praat convention. To avoid a conflict Praat automatically converts the first character of the associated variable to lowercase. In this way the field name, *Frequency*, can start with an uppercase character and the associated variable, *frequency*, with a lowercase character.

The other cosmetic change: the second line is indented to let the form and endform stand out. The first three lines of the script now read as follows.

```
form Play tone
  positive Frequency 377.0
endform
```

C.4.5 Final form

The final improvement is cosmetic again. We want to communicate that the unit for the "Frequency" field is Hz. In this script the name of the field has been changed to "Frequency_(Hz)". Despite this change, the associated variable is still named "frequency". During the creation of the form, Praat chops off the last part "_(Hz)" to create the variable. Actually, Praat chops off "_(" and everything that follows at the end of the field name.

Algorithm 2 The final Play tone example.

```
form Play tone
  positive Frequency_(Hz) 377.0
endform
Create Sound from formula... s Mono 0 0.2 44100
  ... 1/2*sin(2*pi*frequency*x)
Play
Remove
```

C.4.6 Variation

Suppose you want to keep the Sounds that were created by the script. You remove the last line in the script and all the newly created Sounds will be kept in the list of objects. However, they

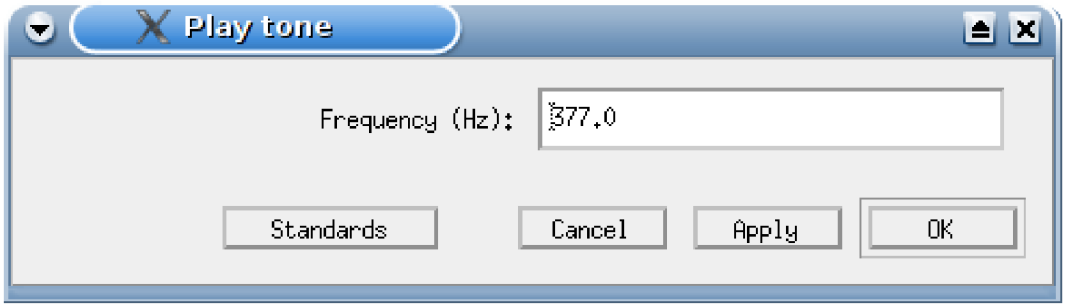

Figure C.8: The final Play tone form.

all carry the same name. You want them to have a meaningful name that enables you to easily identify the Sounds. The new script:

```
form Play tone
  positive Frequency_(Hz) 377.0
endform
Create Sound from formula... s_'frequency' Mono 0 0.2 44100
  ... 1/2*sin(2*pi*frequency*x)
Play
```

All your sounds will have names that start with "s_" with the frequency attached. For example if you run this script and type "1000" in the "Frequency" field, the sound appears with the name "s_1000". The single quotes in the `'frequency'` term instructs that the *value* of the frequency variable has to be substituted and not the variable name.

C.4.7 Formula..., modifying an existing sound

Instead of creating in one step a sound filled from a formula, we can also do it in two separate steps. In the first step we create a silent sound, i.e. a sound filled with zeros. In the second step we modify the contents of the sound with a formula. In the script we modify the "two line" sound creation into the following.

```
Create Sound from formula... s_'frequency' Mono 0 0.2 44100 0
Formula... 1/2*sin(2*pi*frequency*x)
```

Note that we have replaced the formula part in the original script with the formula "0". This creates a new sound with all the samples having the value 0. The line that starts with "Formula..." modifies the contents of a selected Sound object. Because in the line immediately before the formula a new Sound was created, this new Sound will be selected automatically and the formula will be applied. More on Formula... in section C.6.1.1. in the next chapter, we continue with the sript from section C.4.6.

C.5 Conditional expressions

One of the first things you need in a script is to be able to vary its execution path. Sometimes you only want a particular part of the script executed, if a certain condition is fulfilled. For

example, a frequency has to be a positive number, like in the examples of the previous section. Here the positivity condition was automatically maintained by the form and we didn't have to test it explicitly in the script. However, if you happen to fill out a frequency larger than the Nyquist frequency, aliasing occurs and the frequency of the generated tone will not be as you typed. More on aliasing in section B.6.4. If we want to prevent this from happening, two options remain. The first option is to increase the sampling frequency of the Sound in order to faithfully represent the desired frequency. You probably would need a special sound card to create Sounds with a sampling frequency above 44100 Hz. Given the right hardware, *you* would not be able to hear this *ultrasonic* Sound. May be your dog would hear the Sound if the frequency does not exceed 45 kHz. If your tone is even higher, maybe a nearby swimming dolphin could hear it.[8] The other option is: exit the script with an error message. Change lines 4 and 5 in the script and include the following *conditional expression* in the Play tone script:

```
if frequency >= 22050
  exit The frequency must be lower than 22050 Hz.
else
  Create Sound from formula... s_'frequency' Mono 0 0.2 44100
  ... 1/2*sin(2*pi*frequency*x)
endif
```

In this way the generated tone will always have the frequency filled out in the form. A notational variant that would have the same effect is

```
if frequency >= 22050
  exit The frequency must be lower than 22050 Hz.
endif
Create Sound from formula... s_'frequency' Mono 0 0.2 44100
  ... 1/2*sin(2*pi*frequency*x)
```

Frequencies too low for us to hear, say lower than say 30 Hz, are called *infrasonic* frequencies. Elephants use infrasound to communicate. You could extend the script with an extra test for infrasound:

```
if frequency >= 22050
  exit The frequency must be lower than 22050 Hz.
elsif frequency <= 30
  exit Don't pretend to be an elephant.
endif
Create Sound from formula... s_'frequency' Mono 0 0.2 44100
  ... 1/2*sin(2*pi*frequency*x)
```

We can combine tests with "and" and "or" like in the following:

```
if frequency <= 30 or frequency >= 22050
  exit Frequency must be larger than 30 and smaller than 22050.
endif
Create Sound from formula... s_'frequency' Mono 0 0.2 44100
  ... 1/2*sin(2*pi*frequency*x)
```

or in another variant

[8] According to George M. Strain's website on hearing at Louisiana State University (http://www.lsu.edu/deafness/HearingRange.html).

```
if frequency > 30 and frequency < 22050
  Create Sound from formula... s_'frequency' Mono 0 0.2 44100
  ... 1/2*sin(2*pi*frequency*x)
else
  exit The frequency must be higher than 30 Hz and lower than 22050 Hz.
endif
```

For the conditional expression in a formula with commands like `Create Sound from formula...`, we have to use a syntactical variant. Because a formula is essentially a one-liner we use the form

```
if bla then blabla else blablabla fi
```

in which the bla-parts are expressions and the else part is *not* optional. For example, the following one-liner creates a tone with a gap in it (or two tones if you like).

```
Create Sound from formula... gap Mono 0 0.3 44100
  ... if x>0.1 and x <0.2 then 0 else 1/2*sin(2*pi*500*x) fi
```

If you select an interval you can do this by combining the lower limit and upper limit with "and" as the previous script does or with "or" as in following one. Both scripts result in exactly the same sound.

```
Create Sound from formula... gap Mono 0 0.3 44100
  ... if x<=0.1 or x >=0.2 then 1/2*sin(2*pi*500*x) else 0 fi
```

Another variant involves the use of the variable *self*.

```
Create Sound from formula... gap Mono 0 0.3 44100
  ... 1/2*sin(2*pi*500*x)
Formula... if x>0.1 and x <0.2 then 0 else self fi
```

We create a tone first and then modify the existing tone with a formula. In the else part the expression *self* essentially says "leave me alone". The *self* indicates that the else part applies no changes here.

C.5.1 Create a stereo Sound

One of the ways to create a stereo Sound is with the `Create Sound from formula...` command.[9] In this section you will make the same sound in both channels and you will learn how to use a conditional expression to make different sounds in the left and right channel. Also you will learn something about *beats*.[10]

Start by creating a stereo Sound that is a combination of two tones that slightly differ in frequency:

```
Create Sound from formula... s Stereo 0 2 44100
  ... 1/2 * sin(2*pi*500*x) + 1/2 * sin(2*pi*505*x)
Play
```

[9] Another way is to select two mono sounds together and combine them via the Combine Sounds - >Combine to stereo command.

[10] A beat is an interference between two sounds of slightly different frequencies, perceived as periodic variations in volume whose rate is half the difference between the two frequencies.

This command differs from the previous ones in your choice of the "Stereo" option in the "Channels" field. Furthermore, the formula part contains a sum of two tones. Click OK, and the new sound appears in the list of objects. To check that the Sound is stereo, click the "Edit" button in the dynamic menu. If two sounds appear in the editor, one above the other, the selected Sound has two channels and is a stereo file.

Another way to check for stereo is to click the "Info" button at the bottom of the Object window when the Sound is selected. A new "Info" window pops up, showing information about the selected Sound. The info window starts with general information about the Sound.

```
Object type: Sound
Object name: s
Date: Mon Feb 18 20:39:12 2008
Number of channels: 2 (stereo)
Time domain:
   Start time: 0 seconds
   End time: 2 seconds
   Total duration: 2 seconds
Time sampling:
   Number of samples: 88200
   Sampling period: 2.2675736961451248e-05 seconds
   Sampling frequency: 44100 Hz
   First sample centred at: 1.1337868480725639e-05 seconds
```

Figure C.9: The first part of the text in the Info window for a stereo Sound.

On the fourth line the number of channels will show 2, which means that it is a stereo Sound. The next lines show information on the time domain, followed by information on the digital representation of the Sound.

Listen to the Sound. You will hear beats: the sound increases and decreases in intensity. The sound is equal in both channels.

Now create the following new stereo Sound with the following script:

```
Create Sound from formula... s Stereo 0 1 44100
  ... if row=1 then 1/2 * sin(2*pi*500*x) else 1/2 * sin(2*pi*505*x) endif
```

or with the notational variant:

```
Create Sound from formula... s Stereo 0 1 44100
  ... 1/2 * sin(2*pi*(if row=1 then 500 else 505 endif)*x)
```

A stereo Sound in Praat is represented internally as two rows of numbers: the first row of numbers is for the first channel, the second row is for the second channel. The conditional expression in the formula part of the script above, directs the first row (channel 1) to contain a frequency of 500 Hz and the other row (channel 2) to contain a frequency of 505 Hz.[11]

Listen to this Sound but *don't use* your headphones yet. Instead use the stereo speaker(s) from the computer. If everything works out fine, you will hear beats again.

[11] The part `if row=1 then` tests if the predefined variable row equals 1. The equal sign '=' after the 'if' expression is an equality test and is *not* an assignment. For more predefined variables see section E.1.1.

Now use headphones, play the Sound several times but listen to it only with the left ear and then only with the right ear. You will hear tones that differ slightly in frequency. Finally, listen with both ears and you will hear beats. In contrast to the beats in the previous examples, these beats are constructed in your head.

In figure C.10 the difference between the two stereo Sounds we have created in this section becomes very clear. In upper part (a) you see the separate channels of the first stereo Sound. It contains the same frequencies and beats in both channels. In contrast with this, the channels of the last Sound as displayed in part (b) only show two slightly different frequencies in the two channels. No sign of beats here!

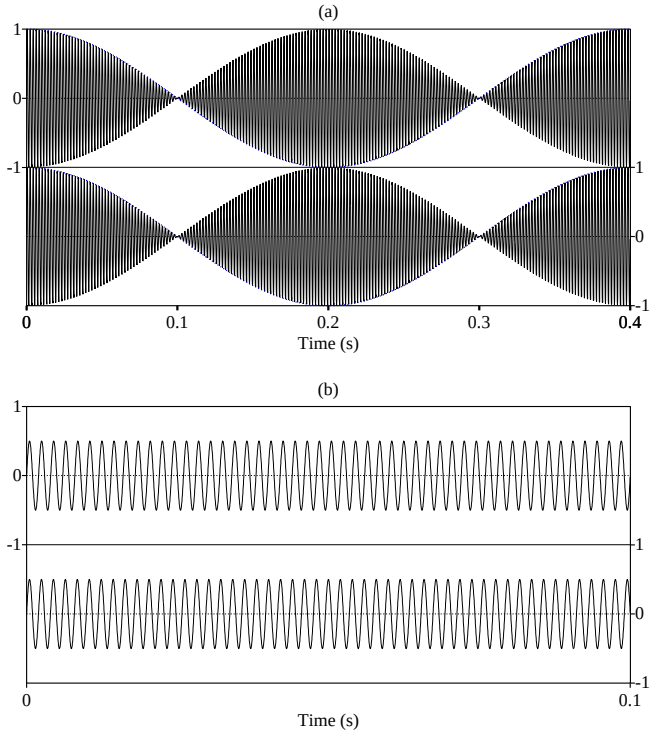

Figure C.10: The stereo channels for the Sounds that (a) have beats in the signal and (b) generate beats in your brain.

C.6 Loops

With a conditional expression you can change the execution path in the script only once. Sometimes you need to repeat an action. In this section we will introduce a number of constructs that enable repetitive series of actions by reusing script lines.

C.6.1 For loops

The following script creates five Sounds with frequencies that increase from 500 Hz to 900 Hz in steps of 100 Hz. The statements between the "for" and the matching "endfor" will be

Algorithm 3 A for loop.

```
for ifreq from 5 to 9
  frequency = ifreq*100
  Create Sound from formula... s_'frequency' Mono 0 0.5 44100
  ... sin(2*pi*frequency*x)
endfor
```

executed exactly 5 times in this script. The logic of this for loop is as follows:

1. At start, the variable *ifreq* is assigned the value 5.
2. Test if *ifreq* is less than or equal to 9. If the test returns *false,* continue after "endfor".
3. The code *inside the loop* is executed
   a) The *frequency* variable is assigned the value ifreq*100, i.e. the *current* value of *ifreq* is multiplied by 100.
   b) A new tone of *frequency* Hz is created. The name of the object will be an "s_" followed by the frequency as a number.
4. The endfor is reached: the value of the variable *ifreq* is increased by 1 and continue with "for" in step 2.

A shorthand notation is possible if the loop variable starts with 1. We skip the "from 5" part. The previous example could also have been written as

```
for ifreq to 5
  frequency = 400+ifreq*100
  Create Sound from formula... s_'frequency' Mono 0 0.5 44100
  ... sin(2*pi*frequency*x)
endfor
```

If the frequencies you have to generate are not related to each other, you can use an *array of variables*:

```
freq1=111
freq2=231
freq3=277
freq4=512
freq5=601
for ifreq to 5
  frequency = freq'ifreq'
  Create Sound from formula... s_'frequency' Mono 0 0.5 44100
  ... sin(2*pi*frequency*x)
endfor
```

C.6.1.1 What goes on in a Formula...

We now try to make explicit what goes on in the formula part of the `Create sound from formula...` command. This one command performs two consecutive actions: it starts by creating a silent Sound, i.e. all the sample values equal 0. In the next action the silent Sound is modified with the formula. In the following script this is shown by the last two lines. Two equal sounds result whose contents is described by the formula 'blablabla'.

```
Create Sound from formula... s Mono 0 0.5 44100 blablabla
# Equivalent to the following two steps
Create Sound from formula... s Mono 0 0.5 44100 0
Formula... blablabla
```

Before we can go on, we first need to know how Sounds are represented in Praat. Internally Sounds are rows of numbers. A mono Sound is one row of numbers, a stereo Sound is two rows of numbers. Each number in a row, i.e. each column, represents the average value of the amplitude of an analog sound in a small time interval, the *sampling period*. The total *duration* of the Sound is the *sampling period multiplied by the number of samples* in the row. To be able to calculate this duration, Praat keeps the necessary extra information, together with the rows of numbers, in the Sound object itself. In the script you have access to this extra information: the number of samples is the predefined variable *nx* and the sampling period is the predefined variable is `dx`. Instead of the duration Praat keeps the start and the end time of a Sound. The variables `xmin` and *xmax* give you access. In this way the duration can be calculated as `xmax-xmin`. To associate a time with the columns in a row we do as follows. The first sample in the Sound is at the midpoint of the first sampling period and, therefore, at a time `xmin+dx/2`. There is a predefined variable associated with the time value of the first sample, *x1*. The second sample will be at a time that lies `dx` from the first sample at `x1+dx`, the third sample will be `dx` further at `x1+2*dx`, et cetera. The last sample in the row will be at time `x1+(nx-1)*dx`.

The big picture now is that the Formula... command is expanded by Praat like this:

```
for col to nx
  x = x1+(col-1)*dx
  self[1,col]= The actual text of formula in Formula... comes here
endfor
```

the `self[1,col]` is the element at position col in the first row.

For example, `Formula... sin(2*pi*500*x)` is internally expanded like

```
for col to nx
  x = x1+(col-1)*dx
  self[1,col] = sin(2*pi*500*x)
endfor
```

For a stereo Sound there is one extra loop for the rows. The number of rows can be read via the predefined `ny` variable.

```
for row to ny
  for col to nx
    x = x1+(col-1)*dx
    self[row,col] = The actual text of formula in Formula... comes here
  endfor
endfor
```

The formula command is more powerful than we have shown here. The next section will show you more.

C.6.1.2 Modify a Matrix with a formula

With Formula... you can modify all data types that have several rows of numbers (matrices). The most important ones are probably Sound, Spectrum and Spectrogram. As we saw in the previous section things make more sense if you are aware of the implicit loops around the formula text. You can do very powerful things with 'self' in a formula.

- Multiply the sound amplitudes with two

```
Formula... self*2
```

- Rectify a sound, i.e. make negative amplitudes positive

```
Modify... if self < 0 then -self else self fi
```

- Square the Sound

```
Formula... self^2
```

- Chop off peaks and valleys

```
Formula... if self < -0.5 then -0.5 else self fi
Formula... if self > 0.5  then  0.5 else self fi
```

- Create white noise

```
Create Sound from formula... white_noise Mono 0 1 44100 0
Formula... randomGauss(0,1)
```

- Create pink noise

```
Create Sound from formula... white_noise Mono 0 1 44100 0
Formula... randomGauss(0,1)
To Spectrum... no
Formula... if x > 100 then self*sqrt(100/x) else 0 fi
To Sound
```

C.6.1.3 Use multiple Sounds in Formula...

Suppose you have two Sounds in the list of objects named "s1" and "s2" and you want to create a third Sound that is the average of the two. There are two ways to accomplish this in Praat. The following examples will show you the difference between calculating *with* interpretation and *without* interpretation.

```
1 Create Sound from formula... s1 Mono 0 1 44100 0
2 Formula... sin(2*pi*500*x)
3 Create Sound from formula... s2 Mono 2 3 44100 0
4 Formula... sin(2*pi*505*x)
5 Create Sound from formula... s3 Mono 0 3 44100 0
6 Formula... (Sound_s1[]+Sound_s2[])/2
7 Create Sound from formula... s4 Mono 0 3 44100 0
8 Formula... (Sound_s1(x)+Sound_s2(x))/2
```

Line 1 creates a new Sound named s1 in the list of objects that will be selected automatically. The Sound starts at time 0 and lasts for 1 second. Line 2 modifies the selected silent Sound by changing it into a tone of 500 Hz. In line 3 a second Sound with a duration of 1 s is created, but now the Sound's starting time is at 2 s. In lines 5 and 7 we create two silent Sounds s3 and s4, both start at 0 s and end at 3 s. The two fundamentally different ways to do the averaging are in lines 6 and 8 in the use of the indexing with [] and ().

1. Let us magnify what happens in line 6 where the formula works on the selected Sound s3:

   ```
   for col to 3*44100
     self[1,col]=(Sound_s1[1,col]+Sound_s2[1,col])/2
   endfor
   ```

   The Sound s3 lasts 3 seconds, therefore the last value for col is 3*44100. The assignment in the loop to `self[1,col]` refers to the element at position `col` in the first row of the *selected* Sound s3. The value assigned is the sum of two terms divided by 2. Each terms refers to a data item that is *not* in the current selected object! The first term, `Sound_s1[1,col]`, refers to the element at position `col` in the first row of a Sound with name s1. The second term refers to an element at the same position but now in a Sound with name s2.
   In a `Formula...`, the syntax Sound_s1[1,col] and Sound_s2[1,col] refer to the element at position `col` in row 1 from Sound s1 and Sound s2, respectively.
   The loop in more detail now. The first time, when col=1, the value from column 1 from Sound s1 is added to the value from column 1 from the Sound s2, averaged and assigned to the first column from the selected Sound s3. Then, for col=2, the second numbers in the rows are averaged and assigned to the second position in the row of s3. This can go on until col reaches `1*44100+1` because then the numbers in the s1 and the s2 sound are finished, (they were each just one second duration). Praat then assigns to the Sounds s1 and s2 zero amplitude outside their domains. Then indexes that are out of scope for a Sound, like 44101 is for s1 and s2, will be valid indexes but have a zero amplitude. In this way, the final second and third seconds of s3 are filled with zeros, i.e. silence. When you listen to the outcome of the formula, Sound s3, you will hear frequency beats just like you did in section C.5.1.

2. In line 8 the Sounds are also summed but now their time domains are taken into account. We magnify what happens.

   ```
   for col to 3*44100
   ```

```
  x = x1 + (col-1)*dx
  self[1,col]=(Sound_s1(x)+Sound_s2(x))/2
endfor
```

In the third line of this script, the two Sounds are queried for their values *at a certain time*. Now the time domains of the corresponding Sounds are used in the calculation. The domains of s1 and s2 are not the same, the domains don't even overlap. Just like in the previous case Praat accepts the non-overlapping domains and assumes the Sounds to be zero amplitude outside their domains. The resulting Sound s4 is now very different from Sound s3.

The difference between the indexing of the Sounds with [] versus () is very important. In indexing with [] the Sounds were treated as a row of amplitude values. Amplitude values *at the same index* were blindly added, irrespective of differences in domains or differences in sampling frequencies.[12] In indexing with () the Sounds are treated as Sounds, amplitude values of the Sounds *at the same time* were added and averaged.

Only if sampling frequencies are equal and Sounds start at the same time, the two methods result in the same output.

C.6.2 Repeat until loops

```
throws = 0
repeat
  eyes = randomInteger(1,6) + randomInteger(1,6)
  throws = throws +1
until eyes = 12
printline It took 'throws' trials to reach 'eyes' with two dice.
```

C.6.3 While loops

Given a number m find n, the nearest power of two, such that $m \leq n$.

```
n = 1
while n < m
  # next line is shorthand for: n = n * 2
  n *= 2
endwhile
```

C.7 The layout of a script

The layout of a script is important for *you*. It enables *you* to see more easily the structure of the script. Layout is not important for Praat: as long as the script text is syntactically correct, Praat will run the script.

[12] Create Sound s2 with a sampling frequency of 22050 Hz instead of 44100 and investigate the difference between the behaviour of [] and () .

You are allowed, for example, to add additional comments in the script text that may describe in your terms what is going on. The more easily you can identify what is going on in a script the more easily you can check the semantic correctness of the script. The following elements may help you to structure the script so your intentions become clear.[13]

- White space, i.e. spaces and tabs. White space at the beginning of a script line is ignored in Praat.[14] You can use whitespace to help you see more easily the structure of your script. For example in conditional expression you should indent every line between the if and the endif. In all the examples we presented, white space was used with this function. See for example the scripts in section C.5. However, see section C.1 for possible pitfalls.

- Besides white space for laying out the structure you can use comments. Comments are lines that start with "#", "!" or ";". Make comments useful, they should not repeat what is already completely clear in the script. The comment in the following script is useless.

```
# add 2 to a
a = a+2
```

- Continuation lines start with three dots (...). Use continuation lines whenever you want to split a long line into several shorter lines.

[13] For masters of the C programming language there is a yearly contest to accomplish exactly the *opposite*. Programmers try to write the most obscure code possible. For some really magnificent examples, see the website of The International Obfuscated C Code Contest at http://www.ioccc.org.

[14] There are other computer languages like Python in which white space is part of the syntax of the language.

D Advanced scripting

D.1 Procedures

In writing larger scripts you will notice that certain blocks of lines are repeated many times at different locations of the script. For example, when you make a series of tones but the frequencies are not related in such a way that a simple "for loop" could do the job. One way to do this in a for loop is by defining arrays of variables like we did in the last part of section C.6.1. In this section we describe another way, *procedures* and introduce *local* variables.

A procedure is a reusable part of a script. Unlike loops, which also contain reusable code, the place where a procedure is defined and the place from which a procedure is called differ. A simple example will explain. If you run the following script you will first hear a 500 Hz tone tone played, followed by a 600 Hz tone and a 750 Hz tone. The first line in the script *calls* the

Algorithm 4 Use of procedures in scripting.

```
1 call play_tone 500
2 call play_tone 600
3 call play_tone 750
4
5 procedure play_tone .freq
6   Create Sound from formula... t Mono 0 0.2 44100 1/2*sin(2*pi*.freq*x)
7   Play
8   Remove
9 endproc
```

procedure *named* play_tone with an *argument* of 500. This results in a tone of 500 Hz being played.

In detail: Line 1 directs that the code be continued at line 5 where the procedure play_tone starts. The *local variable* .freq will be assigned the value 500 and line 6 will be executed. This results in a tone of 500 Hz being created. Lines 7 and 8 will play and then remove the Sound. When the endproc line is reached, the execution of the script will return to the start of line 2. The execution of line 2 will result in the same sequence of code: execution continues at line 5. The local variable .freq will now be assigned the value 600 and execution continues until the endproc line is reached. Then execution will continue at line 3, and the whole cycle starts anew. The effect of the script is identical to the following script.

```
Create Sound from formula... t Mono 0 0.2 44100 1/2*sin(2*pi*500*x)
Play
Remove
Create Sound from formula... t Mono 0 0.2 44100 1/2*sin(2*pi*600*x)
```

```
Play
Remove
Create Sound from formula... t Mono 0 0.2 44100 1/2*sin(2*pi*700*x)
Play
Remove
```

It may be clear that defining a procedure can save a lot of typing: less typing means less possibility for errors to creep in. A procedure is a way to isolate certain portions of the code. You can than test it more easily and thoroughly. In a procedure you can define local variables.

A local variable is, as the naming already suggests, only known within the procedure. If you don't use the dot in front of the name, the variable's scope is global and its value may be changed outside the script, or, your script modifies an outside variable. This may create very undesired side effects.[1] To show you that .freq is a local variable, substitute in the script on the preceding page for the empty line 4 the following line `printline Frequency is '.freq'`. The Info window would only show "Frequency is '.freq"'. No substitution occurred because no variable .ifreq is known outside the procedure.

D.2 Communication outside the script

D.3 Files

[1] Consider for example the following situation. A procedure is called from within a `for ifreq to 10` loop. In the procedure the variable *ifreq* is assigned the number 4. Your script would never stop...

E Scripting syntax

E.1 Variables

Variable names start with a lowercase letter and are case sensitive, i.e. ‘aBc’ and ‘abc’ are not the same variable. String variables end with a ‘$’, numeric variables don’t.

Examples: `length = 17.0`, `text$ = "some words"`

E.1.1 Predefined variables

A number of predefined variables exist.

- Numeric variables:
  macintosh: 1 on macintosh, 0 elsewhere
  windows: 1 on windows, 0 elsewhere
  unix: 1 on unix 0 elsewhere.

- String variables:,
  newline$: the newline character
  tab$: the tab character
  shellDirectory$: the directory you were when you launched praat
  homeDirectory$: your home directory
  preferencesDirectory$: the directory where the Praat preferences are stored
  temporaryDirectory$: the directory available for temporary storage
  defaultDirectory$: the directory where the script resides

- Matrix variables: the following variables can only be used in a Matrix context, i.e. in Formula
  self: the value in the current matrix element
  row, *col*: current row and column number
  xmin, *xmax*: start time and end time
  nx: the number of samples in a row
  dx: the sampling period
  x1: the *x*-value of the first point in a row
  x: the x-value of the current point in a row
  y, *ymin*, *ymax*, *ny, dy*, *y1*: analogous to the x variables (i.e. the column)

- Table variables:

E.2 Conditional expressions

if *expression*$_1$
statements$_1$
[[elsif *expression*$_3$
statements$_3$
[elsif *expression*$_n$
statements$_n$]]
else
statements$_2$]
endif

Examples:

```
if age < 3
  prinline younger than 3
elsif age < 12
  printline Younger than 12
elsif age < 20
  printline Younger than 20
else
  printline Older than 20
endif
```

E.3 Loops

E.3.1 Repeat until loop

repeat
statements
until *expression*

Repeats executing the *statements* between repeat and the matching until line as long as the evaluation of *expression* does not return zero or false.

E.3.2 While loop

while *expression*
statements
endwhile

Repeats executing the *statements* between the while and the matching endwhile as long as the evaluation of *expression* does not return zero or false.

E.3.3 For loop

for *variable* [from *expression*$_1$] to *expression*$_2$
statements

endfor

If $expression_1$ evaluates to 1, the from part between the [and the] can be left out as in:

for *variable* to $expression_2$
statements
endfor

The semantics of a for loop are equivalent to the following while loop:

variable = $expression_1$
while *variable* <= $expression_2$
statements
variable = *variable*+1
endwhile

F Terminology

ADC An Analog to Digital Converter converts an analog electrical signal into a series of numbers.

Aliasing the ambiguity of a sampled signal. See section B.6.4 on analog to digital conversion.

Bandwidth The bandwidth of a sound is the difference between the highest frequency in the sound and the lowest frequency in the sound. The bandwidth of a filter is the difference between the two frequencies where the

Big-endian The big-end is stored first. See endianness.

DAC A Digital to Analog Converter transform a series of numbers to an analog electrical signal.

Endianness. Refers to the way things are ordered in computer memory. An entity that consists of 4 bytes, say the number $0x0A0B0C0D$ in hexadecimal notation is stored in the memory of big-endian hardware in four consecutive bytes with contents $0x0A$, $0x0B$, $0xCA$, and $0x0D$, respectively. In little-endian hardware this 4-byte entity will be stored in four consecutive bytes as $0x0D$, $0xCA$, $0x0B$ and $0x0A$. A vague analogy would be in the representation of dates: yyyy-mm-dd would be big-endian, while dd-mm-yyyy would be little-endian.

Little-endian The little-end is stored first. See endianness.

Nyquist frequency. The bandwidth of a sampled signal. The Nyquist frequency equals half the sampling frequency of the signal. For example, if the sampling frequency is 44100 Hz, the Nyquist frequency is 22050Hz.

Sensitivity of an electronic device is the minimum magnitude of the input signal required to produce a specified output signal. For the microphone input of a soundcard, for example, it is that voltage that provides the maximum allowed voltage that the ADC accepts if the input volume control is set to its maximum. Generally the sensitivity levels are mentioned in the specifications of all audio voltage accepting equipment.

Transducer a device that converts one type of energy to another. A microphone converts acoustic energy to electric energy while the reverse process is accomplished by a speaker. A light bulb is another transducer, it converts electrical energy into light.