Exemplarische Struktur für Praat-Skripte

Im Folgenden wird davon ausgegangen, dass .wav-Files, .TextGrid-Files und das Skript sowie Ergebnisfiles in der selben Directory stehen und dass die .wav- und ihre zugehörigen .TextGrid-Files gleich heißen (von der Endung abgesehen). (Es kann durchaus sinnvoll sein, die Datensätze in unterschiedliche Directories zu organisieren; man muss dann die Skripte entsprechend anpassen.) Das Folgende beschreibt eine mögliche Struktur für ein Praat-Skript, wenn alle Labels aller Files in einer Directory abgearbeitet werden sollen. Wenn man nur einen File bearbeiten will, benötigt man nur Punkte 3 bis 6. Ich führe hier nur die wichtigsten Schritte auf und behandel sie dann im Detail weiter unten. Skriptteile sind im `Courier`-Font gesetzt, wobei Symbolfolgen, die in Praat festgelegt sind, in **blauem Fettdruck**, und frei wählbare Variablen- oder Objekt-Namen in *rotem Kursivdruck* gesetzt sind.

0) Lege Variablen fest, die später im Skript immer wieder gebraucht werden.
1) Lösche Info-Fenster und Ergebnis-Files und schreibe Kopfzeile(n) in Ergebnis-Files
2) Gehe durch alle TextGrid/Sound-Files einer Directory
3) Öffne das Sound-File; falls notwendig, berechne Daten (z.B. Formanten, Pitch, Spektrogramm, etc.) für das gesamte File
4) Gehe durch alle Segmente eines Files
5) Extrahiere gewünschte Daten (z.B. Formanten in der Mitte von Segmenten; mittlerer Pitch eines Segments; Informationen aus einem 'spectral slice') und schreibe die Daten in das Ergebnis-File. Falls notwendig, lösche nicht mehr benötigte Objekte zu einem Segment
6) Nach dem Abarbeiten eines TextGrid/Sound-Files, lösche alle nicht mehr benötigten Objekte dieser Files
7) Nach dem Abarbeiten aller TextGrid/Sound-Files, lösche nicht mehr benötigte Objekte (i.d.R. File-Liste) und informiere Benutzer, wohin die Daten geschrieben wurden

Im Einzelnen:

0) Lege Variablen fest, die später im Skript immer wieder gebraucht werden.

Z.B. kann man den Namen eines Output- oder Input-Files, die Nummer eines Tiers, Grenzfrequenzen zur Pitch- oder Formantberechnung, Segmentlabels, die untersucht werden sollen, etc. vorbesetzen. Damit werden dieselben Werte an allen entsprechenden Stellen im Skript verwendet und man kann so – durch eine Änderung an einer zentralen Stelle – das Skript einfach an anderen Datensätze anpassen.

1) Lösche Info-Fenster und Ergebnis-Files und schreibe Kopfzeile(n) in Ergebnis-Files

Typischerweise wird man einige Zeilen der Form

```
clearinfo
filedelete 'resultfile$'
fileappend 'resultfile$' file'tab$'label'tab$'time'tab$'mean_Hz'tab$'stdev_Hz'newline$'
```

am Anfang seines Skripts haben.

2) Gehe durch alle (.TextGrid-)Files einer Directory

Da Praat kein Kommando der Form `foreach/füralle` kennt, muss man diese Aktion in mehreren Schritten ausführen:

```
① Create Strings as file list... grid_list *.TextGrid
② nr_grid_files = Get number of strings
③ for i_grid_file to nr_grid_files
④   select Strings grid_list
⑤   grid_file$ = Get string... i_grid_file
⑥   Read from file... 'grid_file$'
⑦   file$ = selected$("TextGrid")
⑧   • • • (Punkte 3 bis 6 dieser Beschreibung)
⑨ endfor
```

Im Einzelnen:

① Anlegen eines String-Objekts namens *grid_list*, in dem alle Filenamen mit der Endung "**.**TextGrid" gespeichert werden.

② Festellen, wie viele Strings (= Zeilen/Filenamen) in der *grid_list* existieren und diesen Wert in *nr_grid_files* speichern. (Das Objekt *grid_list* ist in diesem Moment mit Sicherheit in der Objektliste selektiert, da es gerade angelegt wurde.)

③ Das erste, zweite, dritte, …, *nr_grid_files* Element bearbeiten, dabei sich die jeweilige Nummer in *i_grid_file* merken (die vollständige Syntax für diese **for**-Schleife lautet: **for** *i_grid_file* **from** 1 **to** *nr_grid_files*).
– wenn die **for**-Schleife aber bei "1" anfängt, kann man das "**from** 1" weglassen.)

④ In der **for**-Schleife das String-Objekt *grid_list* selektieren – es ist beim ersten Betreten der Schleife zwar selektiert, aber im Schleifendurchlauf können andere Objekte angewählt werden, und dann ist es beim zweiten, dritten, … Durchlauf an dieser Stelle nicht selektiert (Selektieren kostet praktisch keine Rechenzeit).

⑤ Aus dem gerade selektierten String-Objekt *grid_list* den String (= Filenamen) auslesen, auf den die Variable *i_grid_file* zeigt (und da *i_grid_file* die Werte 1, 2, 3,… annimmt, wird gerade der erste, zweite, dritte,… String ausgelesen), und in *grid_file$* speichern. In *grid_file$* steht dann im Schleifendurchlauf der Name des ersten, zweiten, dritten, … .TextGrid-Files.

⑥ Jetzt wird das .TextGrid-File *grid_file$* eingelesen und als TextGrid-Objekt gespeichert.

⑦ In *grid_file$* steht der Name des .TextGrid-Files mit der Endung ".TextGrid". Im Folgenden werden aber die Objekte (Sounds, Spektrogramme, Pitch, etc.) nur über ihren Namen (ohne Endung) angesprochen. Die Funktion ***selected$*** liefert den Namen des momentan angesprochenen Objekts ("TextGrid"), das an dieser Stelle im Skript das gerade eingelesene TextGrid-Objekt ist. Dieser Name wird in *file$* gespeichert, mit dem dann die jeweiligen Sound-, Spectrogram-, Pitch-,… Objekte angesprochen werden können.

⑧ Hier kommt das hin, was mit dem jeweiligen (Sound-, Spectrogram-, Pitch-, …) Objekten geschehen soll (s. Pkt. 5 unten).

⑨ Hier wird die **for**-Schleife beendet; d.h., auf die Variable *i_grid_file* wird "1" aufaddiert, und wenn der neue Wert kleiner als oder gleich *nr_grid_files* ist, dann wird die Schleife erneut vom Beginn der Schleife an (mit dem neuen Wert von *i_grid_file*) durchlaufen. Wenn der Wert von *i_grid_file* aber größer als *nr_grid_files* ist, dann wird die **for**-Schleife verlassen und mit dem nächsten Kommando fortgefahren, das auf das **endfor** folgt.

In diesem Beispiel wurden erst alle .TextGrid-Files gesucht, und diese werden dann abgehandelt (z.B. werden zu ihnen die zugehörigen .wav-Files eingelesen, s. unten). Man kann natürlich auch das String-Objekt auf Basis der .wav-Files anlegen (und davon die weiteren Filenamen ableiten). Man kann auch eine .wav-Liste und eine .TextGrid-Liste anlegen und sie parallel abarbeiten. Dabei kann es jedoch vorkommen, dass beide Listen nicht dieselbe Folge von Filenamen enthalten und das Skript unter Umständen nicht mit einem Fehler abbricht (was es bei obigem Ansatz tun würde).

3) Öffne das Sound-File (falls es benötigt wird) und berechne Daten für das gesamte File

Falls nicht nur Dauern von Segmenten analysiert werden sollen (wofür man nur die TextGrids benötigt) wird man i.d.R. das zugehörige .wav-File einlesen und für das ganze File die Daten berechnen (z.B. Spektrogramm, Pitch, Intensität), die man später (s. Pkt. 5) benötigt. Es werden also im Grunde alle Daten bereitgestellt, zu denen man sich später (im Pkt. 5) zu bestimmten Zeitpunkten die Daten 'herauspickt', die man ganz konkret braucht. Das hier beschriebene Verfahren ist wesentlich effizienter, als zu jedem Zeitpunkt in (Pkt. 5) die Daten immer wieder für das ganze File zu berechnen und nur einen Datenwert daraus zu nehmen.

```
① Read from file... 'file$'.wav
② To Spectrogram... 0.005 high_freq 0.002 20 Gaussian
  To Formant(burg)... 0 nr_formant 5500 0.025 50
  To Pitch... 0 low_freq high_freq
  To Intensity... 100 0 yes
```

Im Einzelnen:

① Einlesen des .wav-Files, das zum gerade eingelesenen TextGrid gehört. (In *file$* steht der Name des gerade eingelesenen TextGrids ohne ".TextGrid" Endung.)

② Hier stehen Beispiele für Berechnungen, die für das Sound-Objekt, das gerade eingelesen wurde und damit selektiert ist, durchgeführt werden sollen. (Oft wird man nur eine Berechnung durchführen.) Die Parameter für die einzelnen Berechnungen sind genau die Zahlen/Strings die erfragt werden, wenn man 'mit der Hand' die Berechnungen mit den Funktionen in der Objekt-Liste in Praat ausführt. In den Beispielen hier werden Variablen verwendet, die man unter Pkt. 0 definiert haben sollte.

To Spectrogram... 0.005 *high_freq* 0.002 20 Gaussian
Aus dem Spektrogramm kann man z.B. mit *spectral slice* (s. Pkt. 5.1) den Center of Gravity, die Standardabweichung, Skewness und Kurtosis des Spektrums zu einem Zeitpunkt berechnen. Diese Werte hängen wesentlich von der obersten Grenzfrequenz ab, weswegen sie hier als Parameter verwendet wird. (Achtung: "Gaussian" ist keine Variable, sondern ein Textstring, der aber nicht in doppelten Anführungszeichen stehen darf!)

To Formant(burg)... 0 *nr_formant* 5500 0.025 50
Die Formanten sollen mit dem (Standard-)Burg Algorithmus analysiert werden. Hier wird man evtl. die Anzahl der Formanten variieren wollen. (Das Berechnen der Formanten involviert das 'downsamplen' des Sound-Objekts, was relativ zeitaufwändig ist. Will man häufiger Formanten von den selben Files berechnen empfiehlt es sich, erst diese Files getrennt 'downzusamplen' (z.B. auf 11000 Hz, wenn man Formanten bis 5500 Hz berechnen will) und dann von diesen Files die Formanten zu berechnen.)

To Pitch... 0 *low_freq high_freq*
Der Pitch wird im Bereich *low_freq* to *high_freq* berechnet.

To Intensity... 100 0 yes
Hier wird die Intensität mit den Standard-Parametern berechnet ("yes" ist wieder ein Text-String und keine Variable!).

4) Gehe durch alle Segmente eines Files

Meist will man Daten aufgrund eines Segmentnamens erheben (z.B. können die Segmentnamen Transkriptionslabels wie "a:" oder "k-VOT" sein). Anders als ein Mensch, der mehr oder weniger 'direkt' ein Segment auf Grund seines Namens findet, muss ein Skript ein Segment nach dem anderen Segment 'anfassen' und feststellen, ob es den gewünschten Segmentnamen hat. Das Praat-Skript muss deswegen alle Segmente durchlaufen, um die 'richtigen' Segmente zu finden (das erscheint einem Menschen sehr ineffizient, aber 'eins-nach-dem-anderen-ansehen' ist für einen Computer eine sehr einfache und schnelle Angelegenheit, die ihn auch nie langweilt). Da Praat kein Kommando der Form `foreach/füralle` kennt, muss man diese Aktion in mehreren Schritten ausführen (ähnlich wie beim Erstellen einer Liste aller Files unter Pkt. 2, aber im Detail doch etwas anders):

```
① select TextGrid 'file$'
② nr_intervals = Get number of intervals... tier
③ for i_interval to nr_intervals
④   select TextGrid 'file$'
⑤   label$ = Get label of interval... tier i_interval
⑥   • • • (Punkte 4 und 5 dieser Beschreibung)
⑦ endfor
```

Im Einzelnen:

① Das TextGrid-Objekt muss angewählt werden, weil zuvor andere Objekte erzeugt oder angewählt wurden.

② Feststellen, wie viele Intervalle in dem (selektierten) TextGrid stehen. Da jedes TextGrid mehrere Tiers haben kann, muss die Nummer des Tiers (hier gespeichert in der Variablen *tier*, die man am Besten am Beginn des Skripts (s. Pkt. 0) definiert) angegeben werden – auch dann, wenn das TextGrid nur einen Tier hat.

③ Das erste, zweite, dritte, …, *nr_intervals* Interval bearbeiten, dabei sich die jeweilige Nummer in *i_interval* merken.

④ In der **for**-Schleife das TextGrid-Objekt *file$* selektieren – es ist beim ersten Betreten der Schleife zwar selektiert (weil es in ① ja grade eben selektiert wurde), aber im Schleifendurchlauf können andere Objekte angewählt werden, und dann ist es beim zweiten, dritten, … Durchlauf an dieser Stelle nicht selektiert.

⑤ Aus dem gerade selektierten TextGrid-Objekt *file$* das Label (= den Namen) des Intervals auslesen, auf den die Variable *i_interval* zeigt (und da *i_interval* die Werte 1, 2, 3,… annimmt, wird gerade der erste, zweite, dritte,… Label ausgelesen), und in *label$* speichern. In *label$* steht dann im Schleifendurchlauf der Name des ersten, zweiten, dritten,… Labels.

⑥ Hier kommt das hin, was mit oder in dem jeweiligen Interval geschehen soll (s. Pkt. 5).

⑦ Hier wird die **for**-Schleife beendet; d.h., auf die Variable *i_interval* wird "1" aufaddiert, und wenn der neue Wert kleiner als oder gleich *nr_intervals* ist, dann wird die Schleife erneut vom Beginn der Schleife an (mit dem neuen Wert von *i_interval*) durchlaufen. Wenn der Wert von *i_interval* aber größer als *nr_intervals* ist, dann wird die **for**-Schleife verlassen und mit dem nächsten Kommando fortgefahren, das auf das **endfor** folgt.

5) Extrahiere gewünschte Daten und schreibe sie weg

Hier werden die Daten aus den bereits berechneten/geöffneten Objekten geholt, die man meist aufgrund des Segment-Namens haben will (z.B. am Anfang, Ende oder in der Mitte eines Segments; für das ganze Segment, etc.). Aus diesem Grunde muss man abtesten, ob das Segment, das gerade durchlaufen wird, den richtigen Namen hat. Hierbei gibt es zwei Ansätze: (a) Entweder man sucht genau nach einem (oder mehreren) Namen, oder (b) man berechnet alles für alle Labels (oder alle Labels, die überhaupt einen Namen haben) und sortiert später in Excel (oder einem anderen geeigneten Programm) die 'richtigen' Daten heraus. Der Ansatz (b) erscheint viele überflüssige Daten zu erzeugen, aber häufig ist das ein effizienteres Verfahren, da man alle möglichen Daten hat und nicht etwas neu berechnen muss (der größte Zeitaufwand steckt im Pkt. 3.2 beim Berechnen der Daten-Objekte). Beide Ansätze (a) und (b) werden hier in jeweils verschiedenen Versionen vorgestellt. Da sehr unterschiedliche Dinge von Skripten berechnet werden können, werden hier als Beispiel drei verschiedene Berechnungen demonstriert:

(5.1) Spektrale Parameter
(5.2) Formanten
(5.3) Pitch

In jedem Punkt werden zur Demonstration verschiedene Version des 'Abtesten' der Segment-Namen mit einem **`if`**-statement verwendet und unterschiedliche Formen der Ausgaben gewählt.

5.1) Berechnung spektraler Parameter

```
① if 'label$' = "f"
②   anfang = Get start point... tier i_interval
③   ende = Get end point... tier i_interval
④   mitte = (anfang + ende) / 2
⑤   select Spectrogram 'file$'
⑥   To Spectrum (slice)... 'mitte'
⑦   cog = Get centre of gravity... 2
⑧   printline Das CoG im File 'file$' von "'label$'" zur Zeit 'mitte:3' ist 'cog:1' Hz.
⑨   Remove
⑩ endif
```

Im Einzelnen:

① Hier wird getestet, ob *label$* identisch zum String "f" ist. Falls dieser Test positiv ausfällt werden die Statements ausgeführt, die auf das **if** folgen, ansonsten wird hinter dem zugehörigen **endif** (hinter ⑩) fortgefahren. Das *f* muss in (doppelten) Gänsefüßchen stehen, weil genau dieser String vorhanden sein muss. (Ein String wie "fuss" in der Variablen *label$* würde hier nicht zu einer 'wahren' Aussage führen – es muss exakt der String f sein.)

② Der Anfangszeitpunkt des Intervals im eingangs festgelegtem *tier* wird für das Interval mit der Nummer *i_interval* (das ja grade den Namen f hat, wie in ① getestet wurde) in der Variablen *anfang* gespeichert.

③ Der Endzeitpunkt des Intervals von *tier* wird für das Interval mit der Nummer *i_interval* in der Variablen *ende* gespeichert.

④ Die Mitte zwischen *anfang* und *ende* wird berechnet (gewissermaßen der Mittelwert von *anfang* und *ende*) wird in der Variablen *mitte* gespeichert.

⑤ Es wird das bereits berechnete (s. Pkt. 3.2) Spectrogram-Objekt angewählt.

⑥ Von dem jetzt (mit Sicherheit selektiertem Spectrogram-Objekt) wird ein 'spectral slice' zum Zeitpunkt *mitte* berechnet. Diese Aktion erzeugt ein neues Objekt vom Typ Spectrum (das auch automatisch selektiert ist).

⑦ Von dem gerade berechneten 'spectral slice' wird die 'centre of gravity' berechnet und in der Variablen *cog* gespeichert (die Funktion **Get centre of gravity...** benötigt den Parameter "2", der hier mitgegeben wird). (Es könnten hier natürlich noch weitere spektrale Parameter, wie Standardabweichung, Kurtosis, etc. berechnet werden.)

⑧ Die Daten werden in dem Info-Fenster von Praat ausgegeben. Es wird aber nicht nur der gerade berechnete *cog* Wert ausgegeben, sondern auch gesagt, in welchem File, von welchem Segment und zu welchem Zeitpunkt dieser Wert berechnet wurde, damit man später diesen Wert auch entsprechend der Aufnahme zuordnen kann.

⑨ Das Spectrum-Objekt ist noch immer selektiert (da kein anderes Objekt erzeugt oder angewählt wurde) und es wird gelöscht, da es nicht mehr benötigt wird.

⑩ Hier endet das **if**-Statement aus ① und das Skript fährt mit der nächsten Zeile fort.

5.2) Berechnung von Formanten

```
① if ('label$' = "a") or ('label$' = "a:") or ('label$' = "i:") or ('label$' = "I")
②   anfang = Get start point... tier i_interval
③   ende = Get end point... tier i_interval
④   mitte = (anfang + ende) / 2
⑤   select Formant 'file$'
⑥   f1 = Get value at time... 1 'mitte' Hertz linear
⑦   f2 = Get value at time... 2 'mitte' Hertz linear
⑧   fileappend 'resultfile$' 'file$''tab$''label$''tab$''mitte:3''tab$''f1:0''tab$''f2:0''newline$'
⑨ endif
```

Im Einzelnen:

① Hier wird getestet, ob *label$* entweder identisch zum String "a" oder "a:" oder "i:" oder "I" ist. Falls dieser Test positiv ausfällt werden die Statements ausgeführt, die auf das **if** folgen, ansonsten wird hinter dem zugehörigen **endif** (hinter ⑨) fortgefahren.

② siehe Pkt. 5.1 ②.

③ siehe Pkt. 5.1 ③.

④ siehe Pkt. 5.1 ④.

⑤ Es wird das bereits berechnete (s. Pkt. 3.2) Formant-Objekt angewählt.

⑥ Aus dem Formant-Objekt wird der 1. Formant zum Zeitpunkt *mitte* extrahiert, wobei die Daten im linearen Hertz-Format gegeben werden sollen. Man muss beachten, dass es sich hier um die Strings "Hertz" und "linear" handelt, die aber hier nicht in Anführungszeichen stehen dürfen!

⑦ Aus dem Formant-Objekt wird der 2. Formant zum Zeitpunkt *mitte* extrahiert. (Es könnten hier natürlich noch weitere Formaten oder deren Bandbreiten etc. erhoben werden.)

⑧ Die Daten werden in den *resultfile$* geschrieben. Es muss beachtet werden, dass das erste Argument (mit Zwischenraum oder Tab vom Folgenden getrennt) von **fileappend** ein Filename sein muss. Wenn dieses File existiert, werden die Daten an das File angehängt. Wenn das File noch nicht existiert, wird es automatisch erzeugt. Wichtig ist auch, dass hier der Filename, wenn er in einer Variablen steht (wie z.B. hier in *resultfile$*), in (einfachen) Anführungszeichen stehen muss. Gibt man nur *resultfile$* ohne Anführungszeichen an, dann wird ein File mit dem Namen "resultfile$" erzeugt (und nicht mit dem Namen, der in der Variablen *resultfile$* steht!).

⑨ Hier endet das **if**-Statement aus ① und das Skript fährt mit der nächsten Zeile fort.

5.3) Berechnung von Pitch Parametern

```
① if label$ <> ""
②   anfang = Get start point... tier i_interval
③   ende = Get end point... tier i_interval
④   select Pitch 'file$'
⑤   f0_mean = Get mean... anfang ende Hertz
⑥   f0_median = Get quantile... anfang ende 0.5 Hertz
⑦   fileappend 'resultfile$' 'file$''tab$''label$''tab$''anfang:3''tab$''f0_mean:1'
       'tab$''f0_median:1''newline$'
⑧ endif
```

Im Einzelnen:

① Hier wird getestet, ob *label$* ungleich dem ‘leeren’ String ist (die beiden doppelten Anführungszeichen folgen direkt aufeinander, d.h. zwischen ihnen steht nichts). Mit anderen Worten: der Skript-Teil hinter dem **if** und vor dem zugehörigen **endif** wird nur betreten, wenn das *label$* irgendetwas (= wenigstens ein Zeichen) enthält.

② siehe Pkt. 5.1 ②.

③ siehe Pkt. 5.1 ③.

④ Es wird das bereits berechnete (s. Pkt. 2.2) Pitch-Objekt angewählt.

⑤ Aus dem Pitch-Objekt wird der Mittelwert von dem Signalstück, das von *anfang* bis *ende* reicht, in Hertz berechnet und der Variablen *f0_mean* zugewiesen. Man beachte, dass `Hertz` hier ein String ist, der als solcher genau so in der Funktion stehen muss und keine (einfachen oder doppelten) Anführungszeichen haben darf (falls dieser String mittels einer Variablen übergeben werden sollte, kann diese mit einfachen oder auch ohne Anführungszeichen dort stehen). Bei *anfang* und *ende* dagegen handelt es sich um (numerische) Variablen: Praat erwartet an diesen Positionen in der Funktion **Get mean...** Zahlen, und da dort die Namen von (existierenden!) Variablen stehen, werden die Werte dieser Variablen als Zahlenwerte eingesetzt. Diese beiden Variablen dürfen in einfachen Anführungszeichen stehen, müssen es aber nicht.

⑥ Aus dem Pitch-Objekt wird das ‘0.5-Quantil’ (= der Median) von dem Signalstück, das von *anfang* bis *ende* reicht, in Hertz berechnet und der Variablen *f0_median* zugewiesen.

⑦ Die Daten werden in den *resultfile$* geschrieben (s. Erläuterungen zu Pkt. 5.2 ⑧). Dieser Ausdruck hier ist recht lang und erstreckt sich in diesem Dokument über zwei Zeilen, muss im Praat-Skript aber in einer (langen) Zeile stehen.

⑧ Hier endet das **if**-Statement und das Skript fährt mit der nächsten Zeile fort.

6) Lösche alle nicht mehr benötigten Objekte zu einem File

Nachdem alle Segmente eines Files abgearbeitet worden sind, sollte man (muss aber nicht) nicht mehr benötigte Objekte löschen, um nicht zu viel unnötigen 'Müll' mitzuschleppen. Diese Aktion geschieht hinter der **for**-Schleife, in der alle Segmente eines Files abgearbeitet werden (es gibt in diesem Beispiel noch eine weitere Schleife, mit der verschiedene Files abgearbeitet werden!).

```
① select Spectrogram 'file$'
② Remove
③ select Sound 'file$'
④ Remove
⑤ select TextGrid 'file$'
⑥ Remove
```

① Es wird das Spektrogram-Objekt mit dem Namen, der in der Variablen *file$* gespeichert ist, angewählt...
② ...und entfernt.
③ Es wird das Sound-Objekt angewählt...
④ ...und entfernt.
⑤ Es wird das TextGrid-Objekt angewählt...
⑥ ...und entfernt.

Falls noch weitere / andere Objekte zu einem File existieren (z.B. Pitch, Formants etc.) können diese analog entfernt werden.

7) Räume am Ende auf und informiere Benutzer

Hinter der **for**-Schleife, die alle Files einer Liste abarbeitet, sollte man aufräumen, und alles löschen, was nicht mehr gebraucht wird. Zusätzlich ist es für den Benutzer eines Skripts hilfreich zu erfahren, dass das Skript fertig ist.

```
① select Strings wav_list
② Remove
③ printline Ich habe fertig und alle Daten nach 'result_file$' geschrieben.
```

① Es wird das Strings-Objekt *wav_list* angewählt (wobei "wav_list" hier genau dieser String ist, und keine Variable)...
② ...und entfernt.
③ Es wird eine (hoffentlich informative) Meldung in das Info-Fenster von Praat geschrieben.

Und damit ist auch das Ende dieser Beschreibung erreicht.