Session 4 (14:00 – 18:00, Thurs.): The production domain

- Textgrids: planning and creating TextGrids.
- Annotating: segmenting and labelling; good practices; some ways to automatize parts of some processes (finding unvoiced, using reliable acoustic landmarks, using forced aligners).
- Extracting data and measuring: reproducibility and efficiency; querying objects (including TextGrids); outputting results to tables and then to standard external files (formatting and encoding).
- Praat picture: the logic behind the canvas, how to script it, some tricks for nicely looking images and some minimum standards for publication.
- Scripting challenges: small production project (recordings will be provided) to practice segmentation and annotation, data extraction and storage, and preparation of figures.

4.1. Textgrids

- TextGrids are an essential part of speech **production studies**. In most cases, if not all, at least some level of segmentation and human intervention will be needed to prepare the way to extract acoustic data with scripts.
- In my experience, when undertaking small projects, researchers normally don't spend much time thinking about TextGrids before creating them, normally because they usually are indeed very straightforward, and because the consequences of **bad planning** aren't really important at that scale.
- However **larger projects** (a proper production study, for example), where a considerable number of hours will be invested in segmenting and labelling, researchers should spend a healthy amount of time thinking their strategies through.

- Before we dive into this subject any further, **what is a TextGrid** and **how does it look**?

```
wav_id = Read from file: "00_sound_1_word_list.wav"
tgd_id = Read from file: "00_sound_1_word_list.TextGrid"
plusObject: wav_id
View & Edit
```

(4.1)

- A TextGrid is a structured object with duration and strata. The duration is always the same than from the Sound object which it complements. Its strata are called tiers and they can be **annotated** (that is: **segmented** and **labelled**).
  - TextGrids can have many tiers, and they can be segmented into as many parts as necessary, and labelled with anything included in the UNICODE standard.
  - Each tier can be declared as an **interval tier** or a **point tier**.
    - The former divides the temporal dimension in sections (intervals), and thus they have a start point and an end point.
    - The latter specifies a single point in time (thus the name), and they are defined by only one temporal dimension: their location.
  - **Interval tiers** are a succession of contiguous intervals, numbered from 1 to the last one. Each interval can be labelled.
  - **Point tiers** look like a time line in which several points have been bookmarked.

They can also be labelled with any text you like.
    - You would use interval tiers for events that have a beginning and end (e.g., segments, words) and point tiers for those that occur in a specific point in time (e.g., minimum intensity point, maximum pitch).
- Let's also **take a look** to the options that become available on the dynamic menu when we select a Sound object and its corresponding TextGrid, and when we select a TextGrid on its own.

4.1.1. Planning and creating

- **Planning your TextGrid**:
    - The first thing you should have very clear before starting to annotate a signal are which are all your **variables of interest**. You should seat and think this very carefully, because once you've started working, restructuring your TextGrid to accommodate more variables could mean doing it all again.
        - Every single variable of interest **needs to be coded** somewhere in your TextGrid, or calculating it should be possible from your TextGrid. Some examples:
            - Mean formant values for the internal 50% duration of vowels depend on vowels being segmented.
            - Maximum pitch velocity change within a phrase depends on the phrase being isolated.
            - The word or nonsense word status for each lexical item should be coded somewhere in your TextGrid, probably on a separate tier with intervals replicating word boundaries.
            - Extracting the expression of a binary variable that indicates whether a target consonant was located after pause requires that you annotate not only the consonant of interest, but the unit immediately before, which might be a segment or a pause.
        - If you have time in your hands, it's better to collect more variables than those you need at the moment. If you collect fewer, you'll have to go back to the annotation process and duplicate efforts.

    - Both for planning the TextGrid itself, but also for your data query, storage and subsequent statistical analyses, you need to clearly identify what I call the **unit of minimum segmentation**.
        - This unit of minimum segmentation is the **smallest unit** that you will segment and hopefully it will coincide with the main topic of interest in your production study (e.g., vowels; syllables, foot group, /l/ and /r/, etc.).
            - Normally, this unit also coincides with your **datum or token**, and it is the unit that should separate **rows** in a dataset shaped in **long data format**.
        - Whenever possible, this unit should be **contained** by all other higher level units in your TextGrid. So if you're interested in /r/, but you also are interested in syllable type and word frequency, you'll need to segment in separate tiers syllables and words, and you'll have to code syllable type and transcribe words.
        - Identifying this unit and having any other higher levels effectively containing it is very important for **data extraction**, because most times you'll query your Textgrid by **iterating through your unit of minimum segmentation**.

        - For example, if you're interested in vowel formants, you'd extract formant data by iterating from vowel to vowel, but not from word to word, because a word might contain several vowels. If a vowel happens to be included in the same word than the next one, it doesn't matter: they will constitute different rows in your dataset and some variables will be repeated (in this case, the variable word).
            - If you iterate from word to word, you'll have to include a loop within that loop to iterate through vowels that are found inside. You're making your scripts more complex, less clear and you'll be wasting computer resources.
    - **Let's see an example**: Let's take another look at the files we inspected earlier, and let's talk about:
        - Variables of interest.
        - Unit of minimum segmentation.
        - Nesting of smaller units within bigger units.
        - Iteration to extract data.
    - We'll look at a TextGrid from a section of a word list elicitation task and at a TextGrid from a section of a semi-guided conversation elicitation task.

- Once you've decide which are your variables of interest, and you've identified the unit of minimum segmentation, you should **plan the structure** of your TextGrid.
- You should worry about the following questions, amongst others:
    - **How many tiers** will you need to include all you variables of interest? How are you going to name your tiers so that they are informative?
    - Which tiers will be **point** tiers and which ones **interval** tiers? Is the distinction relevant for all variables of only for some? If it is irrelevant for some, is there a better default?
    - Will you use one tier to extract more than 1 indexical or acoustic variable?
    - What type of **coding** will you use for indexical variables such as word, type of syllable, position of stress, presence of stress, neighbouring pauses, phonetic variant, etc.?
        - Prepare lists of all variables that require coding and write a comprehensive list of all the **levels for each categorical variable** (variants). Think out of the box, because speech signals seldom agree with theoretical descriptions.
    - Some tips for the planning stage:
        - Place **higher level units above lower level units**. It makes things easier to segment and more logical to look and work with.
        - Create an interval tier at the bottom of all tiers to write **comments**. These comments might eventually make your reconsider your current TextGrid structure, in particular when you bump into the unexpected. They are also an interesting source of spin-off follow-up studies.
- If you have **different data elicitation tasks**, try to plan a TextGrid structure and coding system that is common for all tasks, and that will allow you to extract the same data from all tasks (even if you have to use NA values for one task).

- **Creating a TexytGrid**: Creating a TextGrid is very easy. All you need to do is select a Sound object, and then go to `Annotate - > To TextGrid...`. Then, you'll be requested

to enter all tier names and to specify which one of these are point tiers. If you had a Sound object selected, you can script this process like this:

```
text_grid_id = To TextGrid: "words syllables vowels", "vowels"
```
(4.2)

- **Time to have some fun 4.1**: Open the file “01_textgrid_project.wav”, which contains six Spanish words uttered by a native Chilean Spanish speaker who had a cold that day.
  - The words are: *agendar* /a.xen.ˈdar/ (“to schedule”), *vaporizar* /ba.po.ɾi.ˈsar/ (“to vaporize”), *fotografiar* /fo.to.gɾa.ˈfjar/ (“to photograph”), *cubo* /ˈku.bo/ (“cube”), *corrector* /ko.rek.ˈtor/ (“correction fluid”), *encuadernar* /en.kwa.der.ˈnar/ (“to bind”).
  - Think of a production study that could interest you (at the segmental or suprasegmental level) that uses this sound file, and define your subject of study (== unit of minimal segmentation).
  - Define at least 3 acoustic variables and 3 indexical variables of interest.
  - Devise a plan for the type of TextGrid that you'd need to extract these variables:
    - Define the number of tiers that you'll need and their names.
    - Define which ones will be interval and point tiers.
    - State which variables will be extracted from which tiers and how.
    - Define the codes that you'll need for the indexical variables (e.g., syllable type, vowel, intonation pattern, etc.).
  - Write a script to create and save that TextGrid for that sound in particular..
  - Don't annotate the TextGrid yet.

4.2. Segmenting and labelling

4.2.1. Annotating

- In order to **manually segment and label a TextGrid**, you need to select your Sound object and your TextGrid object in Praat's Objects window and press `View & Edit`. An editor window will prompt and you'll see the waveform, spectrogram (if the zoom is small enough) and TextGrid.
  - To **edit an interval tier** you have to make click with your mouse in the point in time where you want to add a boundary. You'll see that a boundary marker appears and that your TextGrid now has empty boundaries, and that in top of each boundary there is a small circle.
    - Press that **circle** once you've selected the **point in time** where you want your boundary to be. This will add a fixed boundary. When that boundary has been selected, it shows in colour red. When not, it goes blue.
    - You can also select a **section** of the waveform of spectrogram and then press “Enter” or “Ctrl.” + the number of the interval tier where you want to add the interval (“Command” in Mac), for example, “Ctrl. + 1”.
    - Notice that editing an interval tier creates intervals. The **counter** at the right-hand side indicates the selected interval and the total number of intervals for that tier.
    - You can write any text in any interval. You can use text to transcribe or to code variables of interest.

- **Editing point tiers** is very similar. The only difference is that you can only choose points in time and not sections.
  - Select a point in time and press the little circle on top of the boundary marker to fix the boundary.
  - You can also use “Enter” or “Ctrl.” + the numbered of the tier to enter boundaries without using the mouse so much. Again, you can enter text for each point.
- Besides adding interval boundaries and points, you can of course **move** boundaries, **delete** boundaries and **copy** boundaries.
  - To move a boundary you have to press over it with the mouse, hold the click and drag the boundary to its new location (if you press “Shift”, you can move all the boundaries that are aligned vertically at the same time).
  - To delete a boundary you have to select it and, when it is red, you can use “Alt. + Backspace” to remove it (this might be different in Mac).
  - To copy a boundary, you just select one and then press “Ctrl.” + the number of the tier where you want to copy the boundary.

- **Time to have some fun 4.2**: Practice what you've learned above with the first tier of your production TextGrid project. Edit that single tier and enter the indexical information that you want to extract from that tier.

4.2.2. Good practices and approaches

- There are several things that you can do to make your experience annotating more pleasant and hassle-free.
  - First of all, **save**. As we've said before, Praat doesn't save anything unless you explicitly ask it to do so. This means that you could have been working on a TextGrid for hours and then, when something goes wrong, you can loose all that work.
    - To save a TextGrid the easiest way is to press “Ctrl. + S”, and then overwrite your current TextGrid on your local memory. Do this as often as you feel is necessary.
  - **Copy intervals using shortcuts**, not manual: Many times it will happened that the exact same boundaries will be needed in several tiers (e.g., in one tier you transcribe a word, in another tier you flag whether that word is a noun or a verb).
    - Don't copy boundaries manually, because you might move them a bit and then they stop being aligned. Instead, select one interval and then press “Ctrl.” plus the number of the tier where you want the new boundary or interval. You'll save a lot of time by doing this.
    - By the way, **misaligned** boundaries that are supposed to be aligned can be a big problem afterwards.

  - Use **shortcuts** whenever possible (the less mouse, the better). These are the ones I use more frequently, but there are more:
    - Navigating the TextGrid: Hold “Alt.” and use the arrows.
    - Adding boundaries: “Enter”.
    - Deleting selected boundaries: “Alt. + Backspace”.
    - Copying selected boundaries: “Ctrl. + Tier number”
    - Zoom in: “Ctrl. + I” (the vowel).

        - Zoom out: “Ctrl. + O”.
        - Zoom back: “Ctrl. + B”.
        - Zoom to selection: “Ctrl. + N”.
        - Search: “Ctrl. + F”.
        - Play sound or stop: “Tab”.
        - Play visible window: “Shift + Tab”.
        - Interrupt playing: “Esc”.

- **Time to have some fun 4.3**: Practice what you've just learned above with the next two tiers from your TextGrid and production study project. Try to use as many shortcuts as possible and explore different options to do the same thing until you're comfortable segmenting and labelling.

4.2.3. Some ways to automatize parts of some processes

- You can automatize some annotating processes in Praat, although I insist that, even if you manage to automatize most of it, you still need to manually check and correct segmentation and labelling (unless we are talking about big corpora, but this is also debatable).
    - **Finding pauses**: There are a couple of tools in Praat that can help distinguishing sections that are silences from those that are not. These functions work very well, but they rely on acoustic thresholds that the user has to provide, and for better results you'll need to make an informed decision regarding which values to provide (they do work very well if you select the appropriate settings).
        - **Sound – To TextGrid (silences)...**: This function becomes available when you select a Sound object in your Objects window. You have to go to `Annotate - > To TextGrid (silences)...`. A form will prompt and you'll be asked to enter a series of parameters (let's read the documentation right now).
            - This function attempts to separate **voiced** from **voiceless** sections in your signal, which in turn can be used as a way to find silences if you define the arguments just right. For example, you can ask candidates for voiceless segments to be longer than a given duration threshold, effectively avoiding silences from voiceless plosives to be detected as voiceless.
            - You can call this function from a script by writing a line like the following (which assumes that there is a Sound object selected).

```
tgd_id = To TextGrid (silences): 100, 0.0, -25.0, 0.1, 0.1, "silent",
  ... "sounding"
```
(4.3)

        - The previous function creates an Intensity object and then it calls **Intensity – To TextGrid (silences)...**. This is the function that does all the work on the Intensity object. This function becomes available only when an intensity object has been selected. This function asks for a different number of arguments, which are explained in the documentation.
            - To call this function from a script, assuming you have an Intensity object selected, you can write a line like the following:

```
tgd_id = To TextGrid (silences): -25.0, 0.1, 0.05, "silent", "sounding"
```
(4.4)

- **Time to have some fun 4.4**: Write a script that loads the sound "01_textgrid_project.wav" and executes the two functions from above with default values (you'll need to create an Intensity object for the second).
  - How did they perform? Which one did better?
  - Do you think it would save you some time when starting an annotation process?

```
wav_id = Read from file: "../session_3/01_textgrid_project.wav"
intensity_id = To Intensity: 100, 0.0, "yes"
tgd_id = To TextGrid (silences): -25.0, 0.1, 0.05, "silent", "sounding"
selectObject: wav_id
tgd_id = To TextGrid (silences): 100, 0.0, -25.0, 0.1, 0.1, "silent",
  ... "sounding"
```

(4.5)

  - **Using forced aligners**: Forced aligners are collection of programmes and/or scripts to automatically segment and label speech signals. Several have been built and most of them deal only with well-known languages and/or specific type of input signals (e.g., a fixed set of sentences read aloud).
    - While forced aligners sound like a dream come true for production studies, they should be used very **cautiously** and a human inspection of all the results is necessary to ensure that small and big mistakes are corrected, unless we're talking about big corpora again (but you still would pilot and test the aligner to get your settings right).
    - Also, while forced aligners do most of the heavy lifting at early annotation stages, they still require **input from the user** in the form of an orthographic transcription of the content of the signal, separated in manageable chunks.

    - One rather famous forced aligner is **EasyAlign**:
      http://latlcui.unige.ch/phonetique/easyalign.php
      http://latlcui.unige.ch/phonetique/easyalign/plugin_easyalign.zip
      - This forced aligner supports several languages including English, Spanish, French and Portuguese.
      - Of all the forced aligners I know, this is the easiest to use, and it is relatively user friendly.
      - It only works for Windows, unfortunately. In this OS, it can be executed as an independent programme, but it can also be installed in any operative system as a **plugin** (I've included EasyAlign's plugin in the "session_3" folder).

- **Time to have some fun 4.5**: Is there anyone with a Windows machine in the house? Let's try to install and run EasyAlign. Let's see how we fare.

    - Another aligner is **Prosodylab-Aligner** (haven't tested it):
      http://prosodylab.org/tools/aligner/
      https://github.com/prosodylab/Prosodylab-Aligner

    - And **NCSU Forced aligner** (haven't tested it either):
      https://phon.wordpress.ncsu.edu/lab-manual/forced-alignment/

        - Also the the **Penn Phonetics Lab Forced Aligner** (haven't tested it): http://www.ling.upenn.edu/phonetics/old_website_2015/p2fa/

    - **Using reliable acoustic landmarks**: Sometimes you can avoid having to conduct complete manual segmentation if you (a) have a huge corpus, (b) have a reliable and stable acoustic variable or index that can be identified automatically between broadly defined boundaries.
        - For example, if you manage to install and use a forced aligner to conduct a preliminary annotation of long sounds, then you will be able to automatically identify intensity landmarks such as intensity maxima and minima within vowels, and then you can calculate constriction degree intensity correlates such as: intensity ratio, intensity differences, or maximum and minimum velocity of intensity change.
        - The same is true for any other reliable acoustic measure. Of course, your data will be noisy, and you'd have to report that when talking about your results.
        - **Show example** of intensity in a production study.

4.3. Measuring

4.3.1. Some thoughts on measuring

- How to obtain information from Praat's objects is not a trivial decision. For once, researchers want their data to be representative of what they are measuring.
- Also, there is more than one way to obtain, say, formant values using Praat, and thus the chosen method will have an impact on the data itself.
    - Is using default values when building Praat's objects always a good idea?
- Any data collection effort should strive to, at least, stick to the following guideline:
    - Enough must have been done and enough must have been reported about what was done so that **your work can be reproduced** and (hopefully) **your results replicated**.
    - For example, if you obtain pitch values from an **editor window** manually, the value that you get will depend on how much zoom in or out you have on your screen at that particular moment.
        - If you were to try to reproduce your own analysis from the editor window, you'd most likely obtain slightly different values every time, for several reasons, and these fluctuations accumulate and make your results less reliable.
        - **Let's see this**!
    - The appropriate alternative is to annotate a **TextGrid** to fix the places in time where the measurements are going to be made and to obtain all acoustic **values from Praat's object queries**, such as a Pitch object in this case.
- Once your study achieves reproducibility, you can start thinking about **efficiency**.
    - If there is a faster way of doing something, then that's the method you should choose.
    - Obtaining acoustic information from Praat's objects (as opposed to editors) is, besides reproducible, several times faster than what you could do manually.

4.3.2. Querying objects (including TextGrids)

- The fast and safe method to obtain acoustic or indexical information from Praat's objects is to **query them**, store the values obtained in variables and then save them elsewhere in a more permanent format.
  - You can create all sorts of objects in Praat. One typical route is to open or create a Sound object and then create – for instance – TextGrid, Intensity, Pitch and Formant objects for it, as shown below.

```
sn_id = Create Sound from formula: "sn", 1, 0, 1, 44100, "1/2 * sin(2*pi*377*x)"
To TextGrid: "test", ""
selectObject: sn_id
To Intensity: 100, 0, "yes"
selectObject: sn_id
To Pitch: 0, 75, 600
selectObject: sn_id
To Formant (burg): 0, 5, 5500, 0.025, 50
```

(4.6)

  - In large production studies, in which creating a Formant object for a 10 minute long sound file with a sampling frequency of 44100 Hz takes quite a bit, it is a good idea to create the associated objects only once, store them permanently and then load them for queries whenever that is required. Do the same for other large files such as Pitch objects.
  - When you have an object in Praat's Objects window, the dynamic menu will provide you with a menu called `Query`. **Let's take a look at the options** that are provided for each of the five objects that we've created above.
    - All of these commands are available from a **script** as usual. To use them, all you need to do is to define a variable, assign the result of the command or function to the variable, and provide the arguments that the function requires.
    - For the Sound object, for example, you can write a script that gives you the mean intensity in Pascals for the portion between 0.2 and 06 seconds:

```
mean_intensity = Get mean: 1, 0.2, 0.6
```

(4.7)

    - For the TextGrid, you might want to know whether the first tier is an interval tier or a point tier (it provides a boolean flag: 1 if an interval tier, 0 if a point tier):

```
is_interval = Is interval tier: 1
```

(4.8)

    - For the Intensity object, you could be interested in knowing the time where the maximum intensity happens in object:

```
time_of_max = Get time of maximum: 0.0, 0.0, "Parabolic"
```

(4.9)

  - In a real production study, the data extraction is carried out once the **annotation** has been completed. The typical flow of information in script would be:
    - The script **loads** all required objects.
    - The script **queries the TextGrid** and iterates through the units of minimal segmentation to obtain duration landmarks.

    - The script uses these duration information to know where to calculate duration measurements and to **query all other objects** that require a query.
    - The script **stores** each information batch, row by row, into a Table object.
    - Once the iteration is finished, the Table is saved in an **external CSV** or similar file for subsequent statistical analyses.

- **Time to have some fun 4.6**: Use your TextGrid project to create a script that loads your Sound and TextGrid objects, creates at least 1 additional object for them and then queries the objects and prints the results to Praat's Info window.
    - Your TextGrid most have at least 2 tiers.
    - The script hast to query at least 2 indexical variables and 2 acoustic measurements.

```
wav_id = Read from file: "01_textgrid_project.wav"
tgd_id = Read from file: "01_textgrid_project.TextGrid"
n_intervals_t3 = Get number of intervals: 3

selectObject: wav_id
int_id = To Intensity: 100, 0.0, "yes"

writeInfoLine: "i", tab$, "LABEL", tab$, "DURATION", tab$, "INTENSITY"
for i to n_intervals_t3
  selectObject: tgd_id
  current_label$ = Get label of interval: 3, i
  if current_label$ <> ""
    start_point = Get starting point: 3, i
    end_point   = Get end point: 3, i
    duration    = end_point - start_point
    selectObject: int_id
    mean_int    = Get mean: start_point, end_point, "dB"
    appendInfoLine: i, tab$, current_label$, tab$, fixed$(duration, 3),
  ... tab$, fixed$(mean_int, 2)
  endif
endfor
```
(4.10)

4.3.3. Outputting results to tables and then to external files

- Instead of sending data to the Info window, you'd normally send it to a **Table object**, to save that later as a comma-separated CSV file, which can be imported and read by any text editor or any statistical analysis software.
- During the process of iterating through a TextGrid, I suggest that on each iteration you complete and enter a **full row** to your Table.
- To create a Table object, you need call the command `Create Table with column names`, which requires as arguments: the name that the Table object will have, the number of rows, and the name of the columns. Given that normally you don't know in advance the number of rows that your table will have, I suggest always creating the table with 0 rows.

```
table_id = Create Table with column names: "my_table", 0,
  ... "this that these"
```
(4.11)

- Then, for each **iteration**, you can add a row to your table by using `Append row`, and you can populate your Table by adding string or numeric values with the appropriate

modification commands, which will be illustrated below.

- Once the Table object has been fully completed, all that remains is **saving** it (remember our discussions about paths!).

```
wav_id = Read from file: "01_textgrid_project.wav"
tgd_id = Read from file: "01_textgrid_project.TextGrid"
n_intervals_t3 = Get number of intervals: 3

selectObject: wav_id
int_id = To Intensity: 100, 0.0, "yes"
tab_id = Create Table with column names: "tb", 0, "i label duration intensity"

for i to n_intervals_t3
  selectObject: tgd_id
  current_label$ = Get label of interval: 3, i
  if current_label$ <> ""
    start_point = Get starting point: 3, i
    end_point   = Get end point: 3, i
    duration    = end_point - start_point
    selectObject: int_id
    mean_int    = Get mean: start_point, end_point, "dB"
    selectObject: tab_id
    Append row
    n_rows = Get number of rows
    Set numeric value: n_rows, "i",         i
    Set string value:  n_rows, "label",     current_label$
    Set numeric value: n_rows, "duration",  duration
    Set numeric value: n_rows, "intensity", mean_int
  endif
endfor

selectObject: tab_id
Save as comma-separated file: "03_your_new_data.csv"
```

(4.12)

4.4. Praat picture

- Praat has got some neat drawing capabilities, all of them centred around the Praat Picture window (read full documentation).

4.4.1. The logic behind the canvas

- When you open Praat you should be able to see a window called **Praat Picture window**. This window looks rather mysterious, but that fogginess can be quickly dissipated if we realize that the space that we see there equates to a **canvas**.
- The canvas works as a **Cartesian coordinate system** so that each point can be individualized by two coordinates: one from the “x” axis (abscissae) and another from the “y” axis (ordinates).
  - The canvas is measured in inches (sigh...), and scales in both axes show you marks each half inch.
- Although sometimes you'll actually have to draw single points, which only require 2 values to be located in the canvas, most of the time you'll be more interested in drawing things that fit into rectangles (e.g., spectrograms, pitch contours, spectra), and thus a big part of using Praat's Picture window is **selecting regions**, drawing there something, and then adding details and final touches.

- Just to give you a quick feel about Praat's Picture window, let's run the following script (“04_first_drawing.praat”) and see what happens step by step (not that this is a good image!):

```
Create Sound from formula: "sine", 1, 0, 0.01, 44100, "1/2 * sin(2*pi*440*x)"
Erase all
Select outer viewport: 0, 6, 0, 3
Draw: 0, 0, 0, 0, "no", "Curve"
Draw inner box
Marks right: 5, "yes", "yes", "yes"
```

(4.13)

4.4.2. How to script it

- As usual, the first thing we need to do is to ready our “virtual finger” so that we know the **options** that we have at hand in our scripts for drawing. Some of the options will be object dependent, and other options are only available through the Picture window.
  - Let's spend **some minutes taking a look** at the menus and options from the Picture window...

- Now we'll start creating a relatively standard but rather **complex image** piece by piece, using scripting only. We will create an image that will contain a waveform, a spectrogram, intensity contour and a TextGrid that will show the segmentation for a given signal. We will also show the point of maximum intensity with a horizontal line.
- After loading a WAV file (“04_test_sound.wav”) and a TextGrid (“04_test_sound.TextaGrid”) from your “companion_folder_session_4” folder, the first thing we need to do is to decide what are we going to draw. A quick inspection of both files in Praat made us decide that we want to draw all that is contained in the 3rd interval from tier 1.
  - We need to query the TextGrid now to obtain the **boundaries** of this interval, which should look something like this:

```
wav_id = Read from file: "04_test_sound.wav"
tgd_id = Read from file: "04_test_sound.TextGrid"
start  = Get starting point: 1, 3
end    = Get end point: 1, 3
```

(4.14)

- Next, we need to draw our **waveform**. First, we clean the current state of our canvas by using `Erase all` (be careful with this function). Then, we need to have our Sound object selected, we need to select a space in the canvas where to draw by using `Select inner viewport` and then draw it from start to end (without garnish!), and also a box around it:

```
Erase all
selectObject: wav_id
Select inner viewport: 0.7, 5.3, 0.46, 1.1 ; what are these options?
Draw: start, end, 0.0, 0.0, "no", "Curve"  ; and these?
Draw inner box
```

(4.15)

- Our next task is to create a **Spectrogram** object and draw it. We select the region of the canvas where that will happen (immediately below the waveform), we create the Spectrogram object with default settings and then draw it **without garnish**, add

a nice box around it and also 2 marks at the left-hand side of the graph to make the scale explicit:

```
Select inner viewport: 0.7, 5.3, 1.1, 3
To Spectrogram: 0.005, 5000, 0.002, 20, "Gaussian"          ; what about these?
Paint: start, end, 0.0, 0.0, 100, "yes", 50, 6.0, 0.0, "no" ; and this?
Draw inner box
Marks left: 2, "yes", "yes", "no"                           ; and here?
```
(4.16)

- Next, we will create an Intensity object from our Sound object (with default settings). We will query the point of maximum intensity between our temporal boundaries. Then we change the colour of our pen to `Cyan` and the line width to `2.5`, and draw the **intensity contour**. Then we return to the colour `Black` (otherwise everything else will be cyan), reset the line width and add one mark at the right at the point of maximum intensity, including a line that will go across the spectrogram:

```
selectObject: wav_id
To Intensity: 100, 0.0, "yes"                    ; let's
maximum = Get maximum: start, end, "Parabolic"   ; inspect
Cyan
Line width: 2.5
Draw: start, end, 30, 90, "no"                   ; some more
Black
Line width: 1
One mark right: maximum, "yes", "yes", "yes", "" ; options
```
(4.17)

- After all this has been done, we select the **TextGrid** object once more, we remove its first tier, select the area of the canvas where we want to add the tier (this needs to overlap with the waveform and TextGrid areas) and then we draw it, and then a nice box around it.
  - Be prepared to **iterate the positioning** of the TextGrid until you get it perfectly aligned with the spectrogram on the Y axis. It takes some practice, but you can use one script for drawing something with a TextGrid again and again with other files as long as the number of tiers is the same.

```
selectObject: tgd_id
Remove tier: 1
Select inner viewport: 0.7, 5.3, 0.46, 3.63
Draw: start, end, "no", "yes", "no"
Draw inner box
```
(4.19)

- The big finale is to **save our creation** into an external file. First, we adjust the outer selection of our canvas to remove useless white margin. Then, we save the figure using our favourite formats:

```
name$ = "05_second_drawing"
Select outer viewport: 0.25, 5.8, 0.4, 3.7 ; so, what's this?
Save as 300-dpi PNG file: name$ + "_300.png"
Save as 600-dpi PNG file: name$ + "_600.png"
Save as EPS file:         name$ + ".eps"
Save as PDF file:         name$ + ".pdf" ; working right for you?
```
(4.20)

- **Time to have some fun 4.7**: Try the script from above (is in your "companion_folder_session_4" folder). Remember looking at it in Sublime Text to have syntax highlighting. Once you have it running and working, think of ways in which this image could be improved.

- Before we wrap this up, let's talk about **colours**. First, let's take a look at the contents of the menu `Pen`. You can see that Praat allows you to select between 17 predefined colours to use with the pen tools.
  - These same colours can be used anywhere else in Praat's Picture window where a colour can be entered as an argument (e.g., `World > Paint rectangle...`).
- You can see how these **17 colours** look by running the following script:

```
Erase all

# Storing colour codes into string array.
colour$[1]  = "Black"
colour$[2]  = "White"
colour$[3]  = "Red"
colour$[4]  = "Green"
colour$[5]  = "Blue"
colour$[6]  = "Yellow"
colour$[7]  = "Cyan"
colour$[8]  = "Magenta"
colour$[9]  = "Maroon"
colour$[10] = "Lime"
colour$[11] = "Navy"
colour$[12] = "Teal"
colour$[13] = "Purple"
colour$[14] = "Olive"
colour$[15] = "Pink"
colour$[16] = "Silver"
colour$[17] = "Grey"

# Set vertical range for each bin.
y_range = 0.5

# Loop through colours and paint them 1 by 1.
for i to 17
  Select inner viewport: 1, 2, 0 + (y_range * i), y_range + (y_range * i)
  Paint rectangle: colour$[i], 0, 1, 0, 1 ; what are the arguments here?
endfor

# Select section to export if you want.
Select outer viewport: 0.9, 2.1, 0 + (y_range * 1) - 0.1,
  ... y_range + (y_range * 17) + 0.1
```

(4.18)

- While these colours are ok for initial playing and such, they aren't really pretty and are near impossible to combine nicely[1].
- What we need is more **control** to define a colour ourselves, this is when commands like `Pen > Colour...` or similar become useful.
  - When you select this command via clicks, a pop-up shows up and tells you that you can enter the colour in three formats:

[1] Interested in going serious about colours? Take a look at websites such as COLOURlovers and paletton.com. I'm personally a big fan of palettes, and I try to carefully choose a palette appropriate to my needs and stick to it within a project (e.g., oral presentation slides, a poster or a website). Sites like COLOURLovers even allow you to search by tags.

        - By defining it in a scale from 0 to 1 (only for black, white, and all imaginable shades of grey).
        - By using Praat's colour names, that is the 17 we saw above.
        - By entering its three components in an odd **RGB** format, in which the desired proportion of red, green and blue should be entered as numbers between 0 and 1.
    - There is a huge “however”, though: colours aren't really referred to in that format anywhere. Only Praat uses that strange format.
    - Standard formats to refer to colours are:
        - The **hexadecimal** standard (e.g., “#eb6111”), in which red, green and blue components are referred to in hexadecimal scales.
        - The **RGB** standard, in which the components are expressed as three separate numbers from 0 to 255 (e.g., red: 235, green: 97, blue: 17)
        - The **HSV** format, in which the hue, saturation and value of a colour is defined by using degrees (e.g., hue: 22°, saturation: 93°, value: 92°).
    - Given that colour codes are always expressed in standard formats, you'll need to **convert**, say, RGB colours to Praat's native format every time you need to use them, maybe with a script like this:

```
# Converting colour from  RGB to Praat's native format (RGB: 235, 97, 17).
my_red   = (235 * 1) / 255
my_green = (97  * 1) / 255
my_blue  = (17  * 1) / 255

# Drawing something with that colour.
Select inner viewport: 1.2, 2.3, 1, 2
Paint rectangle: "{'my_red', 'my_green', 'my_blue'}", 0, 1, 0, 1
```

(4.19)

    - If only there was a **plugin** to help us entering standard colour formats directly without having to resort to tedious conversions all the time (**plugin_colour**: http://cpran.net/plugins/colour/).
        - Let's take a good look at this!

4.4.3. Tips for nicely looking and publishable images

- When possible, **prefer vectorial formats**: It hardly has to be justified, but vectorial formats have many advantages over raster.
    - Most of the time they use less computer memory.
    - Being vectorial, they don't **pixelate** if you zoom into them: they just keep going. Journals prefer them for online publishing.
    - It is true that raster images act in more predictable ways and that they cause less trouble in some platforms (Windows and Linux), but most software now can deal with EPS, WMF or PDF formats, and Mac does that particularly well.

- **Avoid the superfluous**: Avoid the temptation of garnishing your images. This is a beginner's mistake.
    - Keep text to a minimum; present your scales in a clever, **informative** and **minimalistic** way.
    - Don't include grids, or anything that clutters your image space or that distracts the reader (e.g., multiple lines across the image).

    - As a rule of thumb, avoid using the option `Garnish`, and add scales, ticks, marks and such strategically.

- **Highlight your message**: While avoiding the unnecessary, include the essential and make your image tell one clear **message**. Hopefully, your reader should be able to grasp the essential idea at once.
    - When possible, use colour to highlight what's central to your story.
    - Avoid arrows.

- **The devil is in the details**:
    - Is your TextGrid aligned with the other elements?
    - Are you saving your image with gigantic useless **white space** margin?
    - Would having a larger/smaller dynamic range make your spectrogram look better?
    - Do you really need all that text? Is that font the best one available?

- **Some additional tips**:
    - To make a contour more salient and visible (intensity, pitch, formant...), draw it first very thick and white, on top of your spectrogram, and then add a thinner version in colour on top of the previous one. It will look as if your contour has a white margin that makes it more visible.

```
# Opening sound to play with.
wav_id = Read from file: "04_test_sound.wav"

# Resetting canvas.
Erase all

# Creating Spectrogram object and drawing it.
selectObject: wav_id
spt_id = To Spectrogram: 0.005, 8000, 0.002, 20, "Gaussian"
Select inner viewport: 1.2, 5.8, 1, 3
Paint: 0.12, 0.74, 0, 8000, 100, "yes", 50, 6.0, 0, "no"
Draw inner box

# Creating Intensity object and drawing it (thick white).
selectObject: wav_id
int_id = To Intensity: 100, 0.0, "yes"
Line width: 5
White
Draw: 0.12, 0.74, 0, 100, "no"

# Drawing it again, on top, now in the intended colour.
Line width: 1
Cyan
Draw: 0.12, 0.74, 0, 100, "no"
```

(4.20)

    - Try to only add ticks in places were you want to emphasize something; this is also true for the temporal dimension.

- **Time to have some fun 4.8**: Produce two versions of the same image, using different intervals from the sample files we've been using so far.
    - One figure should be saved in raster formant, have superfluous content, lack a clear message (== it could be about anything) and it should be untidy. Do

garnish this image.
    - The other should be saved in a vectorial format, be clear and to the point, and attention to detail should be paramount.
    - Create a text document and add a table with 1 row and 2 columns. Place the first image on the left cell and the second image on the right-hand cell.
    - Export your text document as PDF and see the results.

4.5. Scripting challenges

- **Time to have some fun 4.9**: In your folder “companion_folder_session_4” you'll find a wav file called “08_scripting_challenge_production.wav”, which contains a series of short sentences read aloud by a native English speaker from somewhere in Scotland.
    - This challenge is rather big, and the programming section might seem daunting at first, but fear not: the idea behind this huge challenge is to have you attempting to tackle a series of tasks that are very similar to real world tasks you'll be facing when dealing with production data.
    - Remember to adopt a programmer mindset when approaching the scripting section of the challenge.
        - Complete each subsection of the scripting challenge on its entirety before moving into the next subsection. Each one depends on the previous one, so ensure that earlier sections work as expected before moving into later sections.
    - I've intentionally ordered the steps of the scripting challenge in a way that should help you organizing your script.

    - **Segmentation and labelling**:
        - Create and save a TextGrid for that Sound object. The TextGrid must have three tiers: one interval tier for a word-level segmentation, one interval tier for a segment level segmentation, and one point tier.
        - Segment every content word from the recording in your first tier, and annotate each word with the corresponding label (e.g., “bird”, “found”, “juicy”, “worm”...).
        - Segment every vowel within a content word in your second level tier. Label each vowel orthographically. These vowels will become your minimum unit of segmentation.

    - **Scripting**. Create and save a script that is able to:
        - Read the two files from the hard disk.
        - Create a Formant object from the Sound object, with default values for all arguments except maximum frequency, which should be 5500 (given that we have a female speaker).
        - Create a Table object with the following column names: “interval”, “word”, “vowel”, “duration”, “F1”, “F2”, “F3”.
        - Iterate through the intervals from the second tier (vowels) of your TextGrid and evaluate which ones have been labelled and which ones not.
        - Only for those intervals that have been labelled, it has to query the point in time where they start and end, their duration, and then calculate the exact

middle point of each vowel.

- Insert a point boundary in the third tier of your TextGrid at the middle point of each vowel, which was calculated in the previous step.
- Use the Formant object to obtain F1, F2 and F3 values for the midpoint of every identified vowel.
- Create a row in the Table object for each vowel and store the number of the current vowel interval (from the seconds tier), the label of the word containing the vowel, the label of the vowel, the duration of the interval from tier 2, the F1 value, the F2 value and the F3 value.
- Save the resulting table with acoustic and indexical data onto the hard drive as a CSV file.
- Programme a figure in Praat Picture window from 1 word of your choosing, which should show the waveform, spectrogram, formant tracks (F1 to F3) on top of the spectrogram and the three tiers from your TextGrid. Use a different colour to show the formant tracking, and save the resulting figure into the hard drive in a raster format (600 dpi PNG) and vectorial format (EPS or PDF).