Session 3 (09:00 – 13:00, Thurs.): Advanced Praat usage

- Peeping into the matrix: Praat objects' attributes; using attributes in formulas.
- Pimp my Praat: changing Praat's default settings, menus and buttons; extending its functionality with plug-ins.
- Talking in computer: calling system commands from within Praat; starting or executing Praat from the command line.
- Scripting challenges: creation and implementation of a dummy (but useful) Praat plug-in.

3.1. Peeping into the matrix

3.1.1. Praat objects' attributes

- Praat's capabilities go way beyond what we've discussed so far. For example, advanced users with both a good knowledge of acoustics and programming, can work on their stimuli for perception experiments by using formulas, sometimes almost exclusively.
- Many of these hacks or tricks rely on the fact that most objects in Praat (if not all) are **matrices**: an array of rows and columns of values.
  - Let's take a look at some objects to check these matrices, in particular, to `Sound`, `Pitch` and `Intensity` objects (this script is also contained in your folder "companion_folder_session_3").

```
# Create Sound object from formula and saving it as raw text file.
sound_id = Create Sound from formula: "sine", 1, 0, 1, 44100, "1/2 *
  ... sin(2*pi*440*x)"
Save as text file: "00_sound_matrix.Sound"

# Creating Pitch object and saving it.
pitch_id = To Pitch: 0.0, 75.0, 600
Save as text file: "00_pitch_matrix.Pitch"

# Creating Intensity object and saving it.
selectObject: sound_id
intensity_id = To Intensity: 100, 0.0, "yes"
Save as text file: "00_intensity_matrix.Intensity"
```

(3.1)

- Each Praat object has a number of formal **attributes**. You can explore these attributes by selecting the object of interest in Praat's Objects window and pressing `Inspect`. This will prompt a pop-up window showing you that object's attributes and the value for each attribute in particular.
- There is a number of attributes that an object may have (documentation here), and what these attributes mean **might change** from object to object.
- Some of the most commonly used attributes are:
  - `xmin`: Start of the time domain for most objects, usually 0, but now always!
  - `xmax`: End of the time domain for most objects, normally its duration, but only if the start is 0!

    - **Time to have some fun 3.1**:
      - How can there be a sound that doesn't start from 0? Let's create one right

now.
    - How can we calculate the duration of a sound that doesn't start from 0 reliably? Scripts that obtain or calculate duration of a sound should take this into account.

- `ncol`: The number of columns in a Table, amongst other objects.
- `nrow`: The number of rows in a Table, amongst other objects.
- `nx`: Number of samples in a Sound object; number of analysis frames in a Pitch, Formant, Intensity and other objects.
- `dx`: Sample period (that is, the time distance between two consecutive samples) in a Sound object, in seconds; the time step between consecutive frames in Pitch, Formant, Intensity and other objects.

- Besides attributes being available in formulas, which is why they are so useful to have, accessing them in a script is **faster** than conducting a normal query on an object. This is certainly relevant for large processes, when there are time constraints.
    - In a script like the one below the first query takes (slightly) longer to compute than the second:

```
sound_id = Create Sound from formula: "sine", 1, 0, 1, 44100, "1/2 *
  ... sin(2*pi*440*x)"
stopwatch
n_samples_1 = Get number of samples
time_1      = stopwatch
stopwatch
n_samples_1 = Object_'sound_id'.nx
time_2      = stopwatch
writeInfoLine: fixed$(time_1, 8), tab$, fixed$(time_2, 8)
```

(3.2)

- To access an object's attributes from within a formula you have two options. You can either use the **object's name** or the objects **ID number**. Given that two objects can have the exact same name, I'd recommend that you always use the ID number.
    - If you want to use an object's name to query its attributes, first you'll have to get the **name** of said object, and for that you can use the function `selected$()`, which requires you to specify the type of the currently selected object between quotation marks:

```
name$ = selected$("Sound")
```

(3.3)

    - To access the attributes using the name, you'll have to type the object's type, and underscore, the object's name, a period and the attribute you want. For a Sound object called “sine”, you'd write:

```
sample_number = Sound_sine.nx
sample_number = Sound_'name$'.nx ; why would this be preferable?
```

(3.4)

    - If you want to access the attributes using the ID number, you first have to make sure to have that ID. Normally you'd store that ID as soon as you create an object, as we've been doing before. However, if you can't do that, you can use `selected()` on the currently selected object:

```
id_number = selected("Sound")
```
(3.5)

- To access the attributes using the **ID number**, you write "Object", then an underscore, the ID number, a period and the attribute you want. For a Sound object with ID number 2, you would write:

```
sample_number = Object_2.nx
sample_number = Object_'id_number'.nx ; what's this? why is it preferable?
```
(3.6)

- **Time to have some fun 3.2**:
  - Create a script that loads the WAV file located in "companion_folder_session_3", called "00_test_sound.wav".
  - Make your script create a TextGrid (with 1 interval tier) and a Pitch object with default settings for the Sound object.
  - Use what you just learned about attributes to find out and store the number of samples (nx), the beginning of the first sample (x1) and the sample period (dx) for the Pitch object.
  - Use that information to modify the TextGrid so that you add an interval every two samples.

```
wav_id        = Read from file: "00_test_sound.wav"
pitch_id      = To Pitch: 0.0, 75.0, 600
n_samples     = Object_'pitch_id'.nx
sample_period = Object_'pitch_id'.dx

selectObject: wav_id
textgrid_id = To TextGrid: "samples", ""
for i to (n_samples / 2)
  Insert boundary: 1, (sample_period * 2) * i
endfor
```
(3.7)

3.1.2. Using formulas

- When talking about **formulas** (full documentation here), two types come to mind. The first one, are mathematical formulas, which you'll need to use many times when you're creating acoustic continua in Praat.
  - We already saw some mathematical functions above. Besides these functions, you should know all operators and their precedence. Ordered by precedence from those at the top to those at the bottom:
    - Negation (`--`) and exponentiation (`^`).
    - Multiplication (`*`) and division (`/`); integer division operators `div` and `mod`.
    - Addition (`+`) and substraction (`-`).
    - Comparison operators: `=`, `<>`, `<`, `>`, `<=` and `>=`.
    - Logical operators (in precedence order): `not`, `and`, and `or`.
  - As always, you can break the precedence of operators by using parentheses.

- The other formulas are more interesting. These correspond to formulas to **create** or to **modify** a Praat's object, and they always require a reference to at least one attribute in order to work.

- You can **create** sounds from formulas. For example, to create a periodic sine wave with a fundamental frequency of 440 Hz, you can use the formula `1/2 * sin (2 * pi * 440 * x)`, as shown in the script below (the first element `1/2` controls the amplitude; change it for `1` for a sound twice as loud).

```
Create Sound from formula: "name", 1, 0, 1, 44100, "1/2 * sin(2*pi*440*x)"
```
(3.8)

  - You can even create a sound with harmonics right out the bat, just by adding two sinusoids together:

```
Create Sound from formula: "name", 1, 0, 1, 44100, "1/2 *
  ... (sin(2*pi*220*x) + (sin(2*pi*440*x)))"
```
(3.9)

- You can also **modify** existing sounds by using formulas. One of the simpler examples would be to increase the amplitude of a sound by multiplying itself by a factor, as shown below:

```
Create Sound from formula: "name", 1, 0, 1, 44100, "1/2 * sin(2*pi*440*x)"
Formula: "self * 3"
```
(3.10)

  - `self` is used to refer to the existing content of the Object, so in the case above, to the Sound object.
  - Adding two sounds with a formula is shown below. Notice that we use the last object and refer to the values in the first column (the one storing the amplitude values) by using `col`. Then to refer to the second object we do it as we did when querying attributes, in this case using the name of the object and its own column of amplitude values.

```
Create Sound from formula: "old", 1, 0, 1, 44100, "1/2 * sin(2*pi*220*x)"
Create Sound from formula: "new", 1, 0, 1, 44100, "1/2 * sin(2*pi*440*x)"
Formula: "self[col] + Sound_old[col]"
```
(3.11)

  - If you want to use ID number you can do it like this:

```
id_1 = Create Sound from formula: "old", 1, 0, 1, 44100, "1/2 *
  ... sin(2*pi*220*x)"
Create Sound from formula: "new", 1, 0, 1, 44100, "1/2 * sin(2*pi*440*x)"
Formula: "self[col] + Object_'id_1'[col]"
```
(3.12)

  - You can also subtract one sound from another:

```
Formula: "self[col] - Sound_old[col]"
```
(3.13)

  - Or use inline conditional statements to modify objects:

```
Create Sound from formula: "name", 1, 0, 1, 44100, "1/2 * sin(2*pi*200*x)"
Formula: "if self[col] > 0.1 then 0.1 else self[col] fi"
```
(3.14)

- **Time to have some fun 3.3**: Write a script that creates a Sound from formula, queries some aspect from the sound (acoustic or an attribute), uses mathematical formulas and at least 1 comparison operator to run some calculations on that aspect somewhere in your script, to then apply the result from them into a copy of your original sound (check `Copy`) by using `Formula`.
  - It doesn't really matter what you do to your original sound as long as you use what's required above.
  - In order to use comparison operators, you'll need to run either some type of loop or a conditional jump.

```
s1  = Create Sound from formula: "a", 1, 0, 1, 44100, "1/2 * sin(2*pi*200*x)"
max = Get maximum: 0, 1, "Sinc70"
thr = (max / 10) ; threshold
s2  = Copy: "b"
Formula: "if self[col] > 'thr' then randomUniform(-0.1, 0.1) else self[col] fi"
```
(3.15)

3.2. Pimp my Praat

3.2.1. Changing Praat's default settings, menus and buttons

- The first thing that you might want to know about Praat is that there is a large number of "**buttons**" (commands) that aren't available by default, but that you can inspect and enable by going to the ButtonEditor, which can be found in `Praat > Preferences > Buttons`.
  - It is a good idea to go there from time to time and realize the interesting number of commands that exist that you can't see by default. For example, there is a very useful command called `Fade in` and another one called `Record Sound (fixed time)`.
  - To enable or disable one of these buttons all you have to do is to click on top of the flag variable that indicates whether they are **shown** or **hidden** and that's it!
  - **Let's take a look** at some of them.

- Other default settings that you may want to explore and perhaps change in Praat, by going into `Praat > Preferences` are your `Text reading preferences...` and also your `Text writing preferences...`. You might be interested in modifying your `Sound recording preferences...` and your `LongSound preferences...` as well.

3.2.2. Extending Praat's functionality with plugins

- You can extend Praat's functionality by adding your own **plugins**, which will then appear as new menus or buttons and execute your own scripts (documentation here).
  - Plugins have to be located inside Praat's **preferences directory**, and this directory varies depending on your operative system. Remember that you can find out the location of your preferences directory by using `preferencesDirectory$`.

```
appendInfoLine: preferencesDirectory$
```
(3.16)

- Once you find your preferences directory, create a folder there that starts (literally) with the string "**plugin_**" and then the name of your plugin.
- For example, we will create a silly plugin which will add two commands to Praat: one to create a Sound from formula with default values and then it will chop it in half, and another one to take any existing Sound object and chop it in half.
  - We will call our plugin "**plugin_chopper**".
- Inside your newly created plugin folder, you need to create a file called "**setup.praat**", and inside that folder you will have to specify all the commands that you want add to the fixed and dynamic menus.
  - The **fixed menu** is the one that never changes and resides on top of Praat's Objects window. The **dynamic menu** is the one that changes depending on the object(s) that are selected at a given moment in Praat's Object window.
  - To add a command to your fixed menu you'll need to include a line into your "setup.praat" which starts with the command `Add menu command`, and that then follows the same conventions to add permanent buttons to the fixed menu in Praat which don't depend from plugins (documentation here).
    - By the way, the buttons included via plugins aren't permanent: they get loaded each time you start Praat.
    - The conventions mentioned above are as follows (this was directly taken from Praat's documentation):

Window	The name of the window ("Objects" or "Picture") that contains the menu that you want to change.
Menu	The title of the menu that you want to change. If window is "Objects", you can specify the Praat, New, Open, Help, Goodies, Preferences, or Technical menu (for the Save menu, which depends on the objects selected, you would use Add to dynamic menu... instead). If window is "Picture", you can specify the File, Edit, Margins, World, Select, Pen, Font, or Help menu.
Command	The title of the new menu button. To get a separator line instead of a command text, you specify a unique string that starts with a hyphen ('-'); the ButtonEditor contains many examples of this.
After command	A button title in the menu or submenu after which you want your new button to be inserted. If you specify the empty string (""), your button will be put in the main menu.
Depth	0 if you want your button in the main menu, 1 if you want it in a submenu.
Script	The full path name of the script to invoke. If you saved the script you are editing, its name will already have been filled in here. If you do not specify a script, you will get a cascading menu title instead.

    - And, in our case, the line for our fixed menu command should look something like the line below. Notice that the button will be created inside `New > Sound`.

```
Add menu command: "Objects", "New", "Create & chop", "Sound", 1,
  ... "newChop.praat"
```
(3.17)

    - The praat script which was refered in the line above, `newChop.praat`, had to be stored as a separate file in the plugin folder, and it looks like this:

```
id     = Create Sound from formula: "a", 1, 0, 1, 44100, "1/2 * sin(2*pi*440*x)"
start  = Object_'id'.xmin
end    = Object_'id'.xmax
half   = (end - start) / 2
part_1 = Extract part: start, half, "rectangular", 1, "no"
selectObject: id
part_2 = Extract part: half, end, "rectangular", 1, "no"
plusObject: part_1
```
(3.18)

- To add a command to your **dynamic menu** (documentation here), you need to follow a very similar route. Below, I'm copying the part of Praat's documentation which is relevant to understand the line that you'll have to add to your “setup.praat” file:

Class 1	The name of the class of the object to be selected. For instance, if a button should only appear if the user selects a Sound, this would be "Sound".
Number 1	The number of objects of *class1* that have to be selected. For most built-in commands, this number is unspecified (0).
Class 2	The name of the class of the second object to be selected, different from *class1*. Normally the empty string ("").
Number 2	The number of selected objects of *class2*.
Class 3	The name of the class of the third object to be selected, different from *class1* and *class2*. Normally the empty string ("").
Number 3	The number of selected objects of *class3*.
Command	The title of the new command button (or label, or submenu title). To get a separator line instead of a command text (only in a submenu), you specify a unique string that starts with a hyphen ('-').
After command	A button title in the dynamic menu or submenu where you want your new button. If you specify the empty string (""), your button will be put at the bottom. You can specify a push button, a label (subheader), or a cascade button (submenu title) here.
Depth	0 if you want your button in the main menu, 1 if you want it in a submenu.
Script	The full path name of the script to invoke. If you saved the script you are editing, its name will already have been filled in here. If you do not specify a script, you will get a separating label or cascading menu title instead, depending on the depth of the following command.

  - In our case, we will be adding the command that controls the script that chops any Sound object in half to the dynamic menu, inside `Convert`, because there is were a command like this sort of belongs amongst friends. That line that goes into “setup.praat” should look like this:

```
Add action command: "Sound", 1, "", 0, "", 0, "Chop this", "Convert -", 1,
  ... "chopInHalf.praat"
```
(3.19)

  - Notice that this button will only appear when 1 (and only 1) Sound object has been selected, which prevents this script from failing.
  - This line refers to a script (`chopInHalf.praat`). This script requires a Sound object to do the following:

```
id     = selected("Sound") ; Obtain the ID from the selected Sound.
```
(3.20)

```
start  = Object_'id'.xmin
end    = Object_'id'.xmax
half   = (end - start) / 2
part_1 = Extract part: start, half, "rectangular", 1, "no"
selectObject: id
part_2 = Extract part: half, end, "rectangular", 1, "no"
plusObject: part_1
```

- You would normally **include plugins** when you have tools that you will be using frequently, for example for a project.
- Creating and distributing plugins is a very good way in which you, a proficient Praat programmer, can share scripts with users that don't know what they are doing. All you need your user to do is to manage to copy your plugin into their preferences folder (you could even write a Praat script that does this for them! All you would need is some patience and `runScript` (read the documentation for plugins in Praat, linked above)).
  - This idea of using plugins to expand the capabilities of Praat is not new, and there are actually ongoing projects to centralize and standardize Praat add-ons just like you can do it when loading packages for *R* or when using libraries for Perl or Python.
  - I'm of course talking about CPrAN – The plugin manager for Praat, led by the resourceful José Joaquín Atria.

- **Time to have some fun 3.4**: Your first challenge will be to install the plugin detailed above in the computer you're using today. Naturally, this will only work if you have clearance to access your preferences folder.
  - All you have to do is to (a) find the preferences folder; (b) create your "setup.praat" and script files; (c) copy the lines from above where they belong and save these files; and (d) reset Praat.

- **Time to have some fun 3.5**: Add a new command to the dynamic menu that chops any provided Sound objects in as many parts as the user wants. You'll have to create a new script that handles this and modify the files to add a new command somewhere.

```
Add action command: "Sound", 1, "", 0, "", 0, "Chop this", "Convert -", 1,
  ... "chopInHalf.praat"
```

(3.21)

```
form
  comment Please, enter the number of pieces:
  integer Pieces:
endform

id    = selected("Sound")
name$ = selected$("Sound")
start = Object_'id'.xmin
end   = Object_'id'.xmax
piece = (end - start) / pieces

for i to pieces
```

(3.22)

```
  selectObject: id
  current_start = start + (piece * (i - 1))
  current_end   = start + (piece * i)
  part_id[i] = Extract part: current_start, current_end, "rectangular", 1, "no"
  Rename: name$ + "_part_" + string$(i)
endfor

nocheck selectObject: undefined
for i to pieces
  plusObject: part_id[i]
endfor
```

3.3. Talking in computer

3.3.1. Calling system commands from within Praat

- It should be no surprise by now that you can also call system commands from within a Praat script (documentation here). If you're a command line fan, then this is what you were waiting for.
  - System commands conventions vary from platform to platform, so bear that in mind when using these commands.
  - All you need to do to run a system command is using the function `runSystem` and then use commas to separate the elements in your call.
  - The following script, which expands the example from the documentation, should suffice to get you started (this example should run fine in Linux and Mac):

```
directory$ = "storage"
Create Sound from formula: "a", 1, 0, 1, 44100, "1/2 * sin(2*pi*440*x)"
Save as WAV file: directory$ + "/test_sound.wav"

pauseScript: "Check the contents of the folder STORAGE now."

runSystem: "rm ", directory$, "/*.wav"

pauseScript: "Check the contents of the folder STORAGE again."
```
(3.23)

  - Notice that your command line is actually receiving the line:

```
rm storage/*.wav
```
(3.24)

3.3.2. Starting or executing Praat from the command line

- More interestingly, you can use Praat via the command (documentation here). Some of the (in my opinion) least interesting options are just **opening Praat** or opening Praat **with some objects** or **a script**. More interesting options are:
  - Asking Praat to **run a script**: To do this, you need to specify the full path to the executable Praat file (which is platform dependent), add use `--run` and then the name of your script enclosed in double quotation marks (plus its full path if you're not currently in the directory where the script is located). The lines that you'd need to use in Linux and Mac look more or less like this, respectively:

```
# If you installed Praat by using "sudo apt-get...":
/usr/local/bin/praat --run "02_system_call_1.praat"

# In my particular case:
/home/mauricio/programming/praat_source/praat --run "02_system_call_1.praat"
```
(3.25)

```
/Applications/Praat.app/Contents/MacOS/Praat --run "02_system_call_1.praat"
```
(3.26)

- The script that was executed ("02_system_call_1.praat") contained:

```
writeInfoLine: "Starting script."
Create SpeechSynthesizer: "English_rp", "default"
To Sound: "This script has been summoned from the system command.", "no"
Play
appendInfoLine: "Terminating script."
```
(3.27)

- Asking Praat to **run a script with arguments**: In order to do this, you need to have a script that takes arguments via a form. Then, in the system call, all you need to do is to provide these arguments one by one after the script has been called, separated by spaces and with strings in single quotation marks and numbers as numbers:

```
/home/mauricio/programming/praat_source/praat --run "02_system_call_2.praat"
"LOL"
```
(3.28)

```
/Applications/Praat.app/Contents/MacOS/Praat --run "02_system_call_2.praat" "LOL"
```
(3.29)

- The script that has been called in the lines from above are contains the following:

```
form Enter arguments.
  sentence Your_message:
endform

Create SpeechSynthesizer: "English_rp", "default"
To Sound: your_message$, "no"
Play
```
(3.30)

- **Time to have some fun 3.6**: Create a script that adds the number 142857 to itself a number of times defined by the user and prints the result of each iteration to the screen. Call this script from the command line.

```
form Enter how many iterations
  integer Iterations:
endform

number = 142857
for i to iterations
  number += number
  appendInfoLine: i, tab$, number
endfor
```
(3.31)

3.4. Scripting challenges

- **Time to have some fun 3.7**: Create any of the following plugins for Praat (they have been ranked by difficulty level):
  - **plugin_reverse**: A plugin that takes a string input from the user, reverses it (e.g., from “tomato” to “otamot”) and displays the result in Praat's Info window. You're going to need string functions to make this happen.
    - Additional challenge: Make your script detect and report whether the result of inverting a string is the same as the original string, essentially, program a palindrome detector (e.g., the reverse of “racecar” is “racecar”).
  - **plugin_countdown**: A plugin that takes a numerical input from the user and generates a countdown from that number to 0 that is played one by one by Praat's synthesizer (the synthesizer has to produce the words, as in “ten..., nine..., eight...”). Make sure to evaluate that the number is legal for a countdown (i.e., that is a positive integer).
    - Challenge for intermediate scriptors: Make the script tolerate negative integers as well, in which case it should perform a countdown from negative to 0.
  - **plugin_singaloud**: A plugin that takes an input from a CSV file containing the musical notes from a song (in Hz) and their duration (in seconds), and makes Praat sing said song using sinusoids. You can find the data you need in the file “03_happy_birthday.csv”, contained in the folder “companion_folder_session_3”. You'll have to use `Read Table from comma-separated file` to import the CSV file into Praat.
    - Challenge for advanced users: Instead of having Praat just singing the song, have Praat create a unique sound file containing the sinusoid song along with a TextGrid segmented and labelled with the syllables corresponding to each note.
    - Challenge for really advanced users: Use Praat's synthesizer to create all the syllables that you'll need to sing happy birthday, and modify their duration and Pitch by using `Manipulate - > To Manipulation...`. Once they have been modified, concatenate them and have Praat sing happy birthday more or less properly.
      - In order to manipulate duration, you'll have to calculate by which factor you'll have to multiply the current duration of the syllable so that it reaches the desired duration.