Praat Scripting

Part 1: Introduction

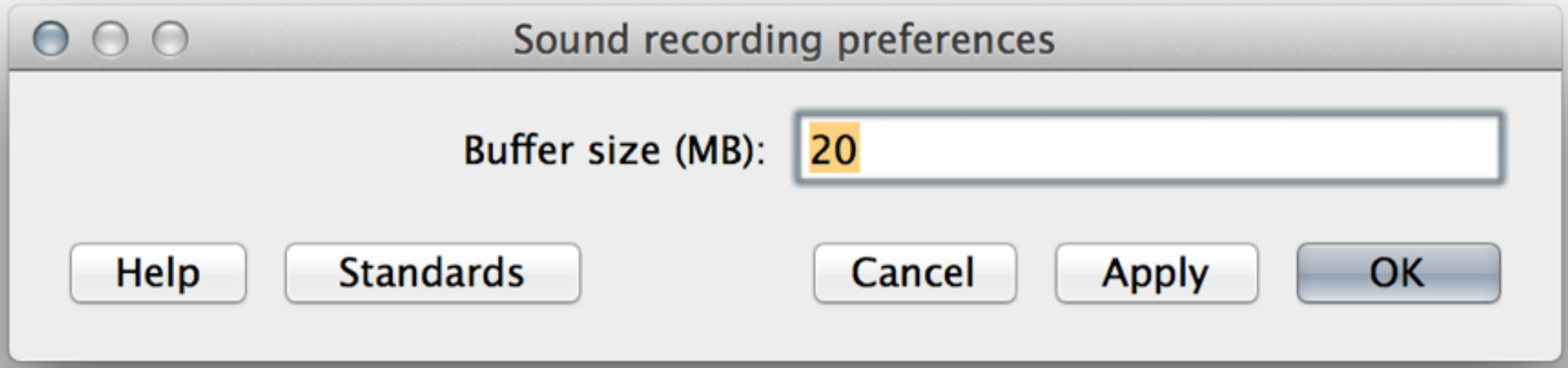
Praat Scripting

Part 1: Introduction

Praat Scripting

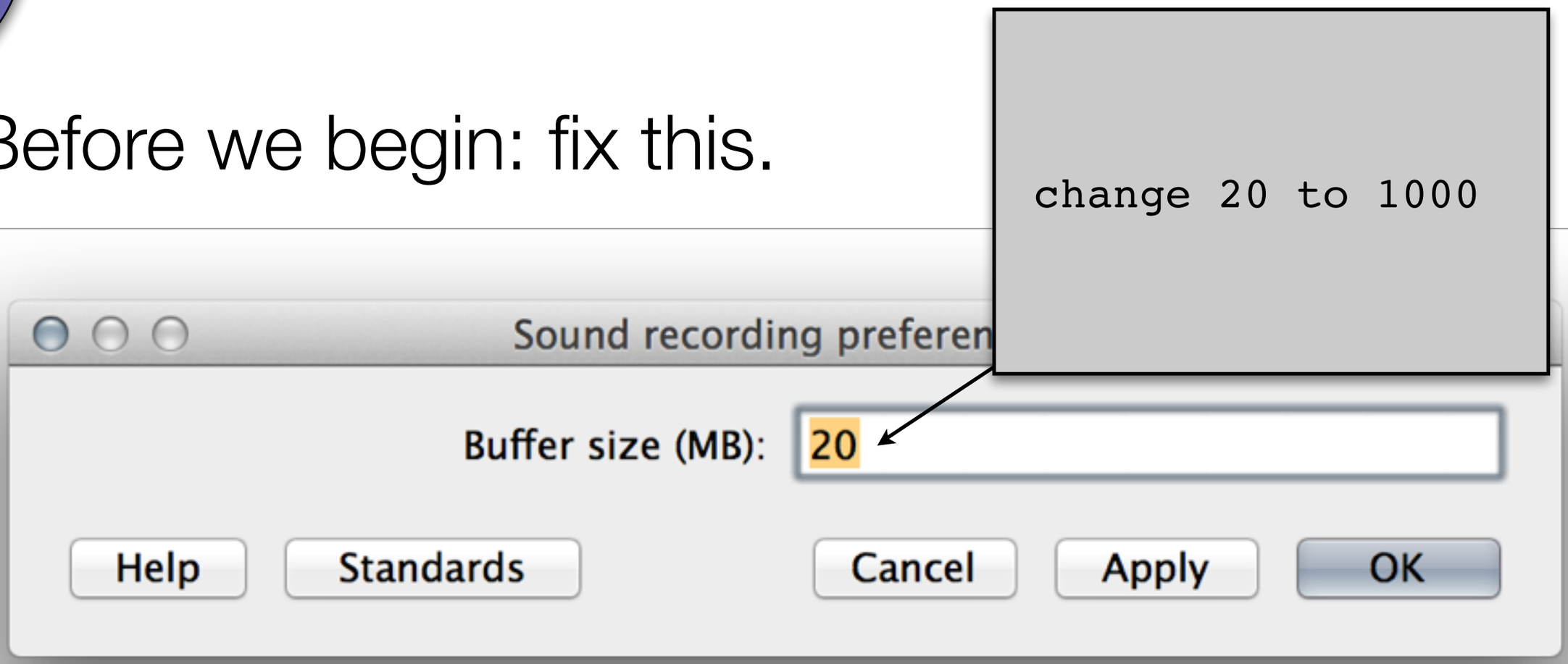
Part 1: Introduction

Before we begin: fix this.



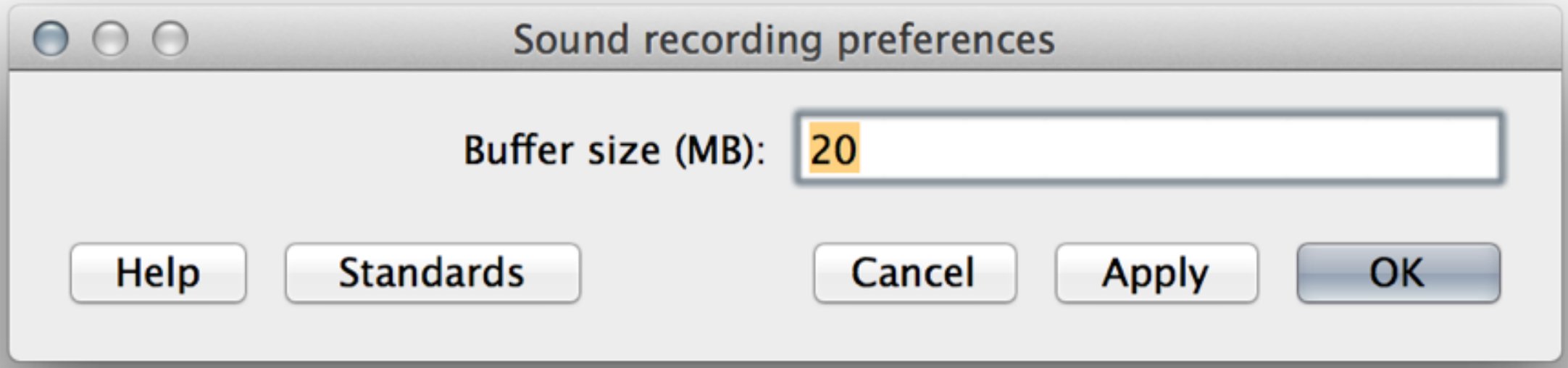
- Praat > Preferences > Sound Recording Preferences
- 20 mb buffer gives you < 4 minutes of recording time (assuming 16 bit/44.1 kHz)
- 1000 mb gives you 3.3 hours of recording time.

Before we begin: fix this.



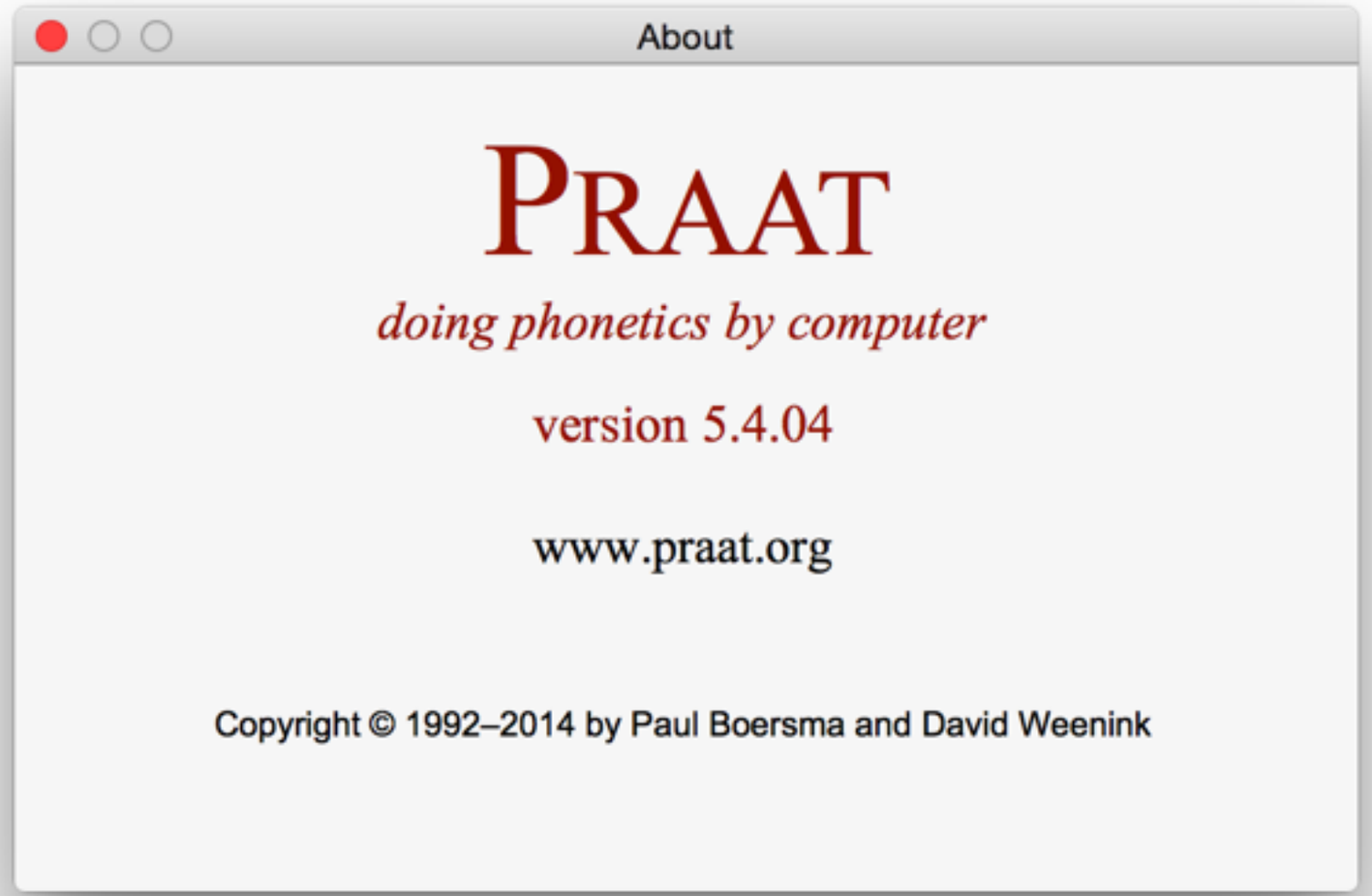
- Praat > Preferences > Sound Recording Preferences
- 20 mb buffer gives you < 4 minutes of recording time (assuming 16 bit/44.1 kHz)
- 1000 mb gives you 3.3 hours of recording time.

Before we begin: fix this.



- Praat > Preferences > Sound Recording Preferences
- 20 mb buffer gives you < 4 minutes of recording time (assuming 16 bit/44.1 kHz)
- 1000 mb gives you 3.3 hours of recording time.

Before we begin...



- Please launch Praat and check your version number (Praat → About Praat...)

- Then check the Praat home page for the current latest version and see what's new

- The scripting language is under active development lately, so many commands will not work in old versions.

Citing Praat

- It is standard practice to include version number and download date when citing Praat in published work.

- Be sure to make a note of these when upgrading.

Boersma, Paul & Weenink, David (2014). Praat: doing phonetics by computer [Computer program]. Version 5.4.04, retrieved 29 December 2014 from http://www.praat.org/

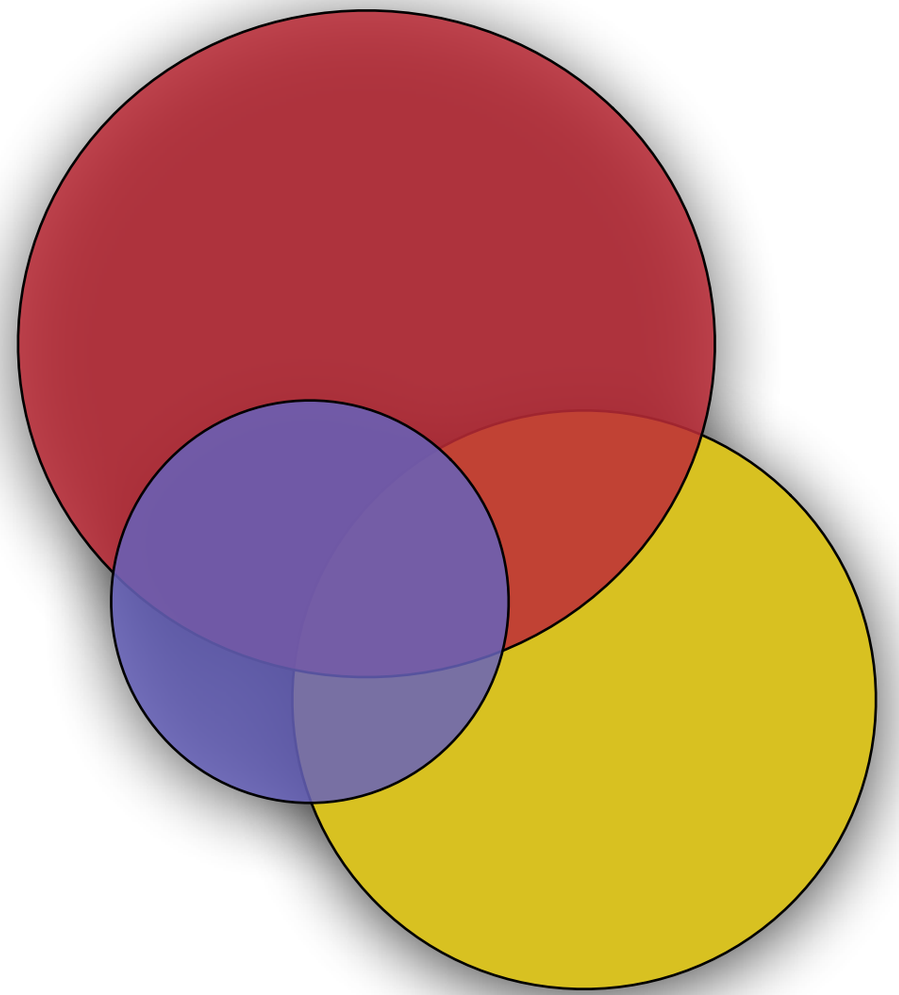
Mailing List

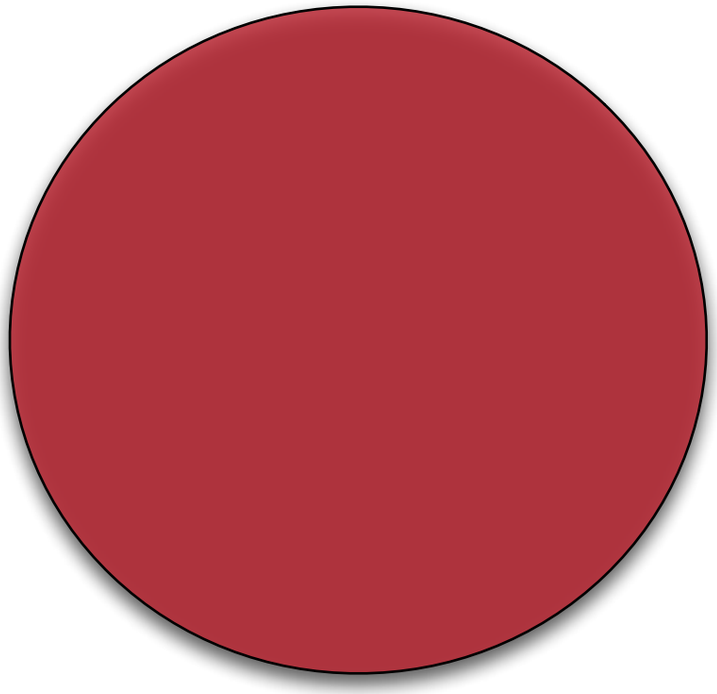
http://uk.groups.yahoo.com/group/praat-users/

praat-users@yahoogroups.co.uk

What exactly is our task?

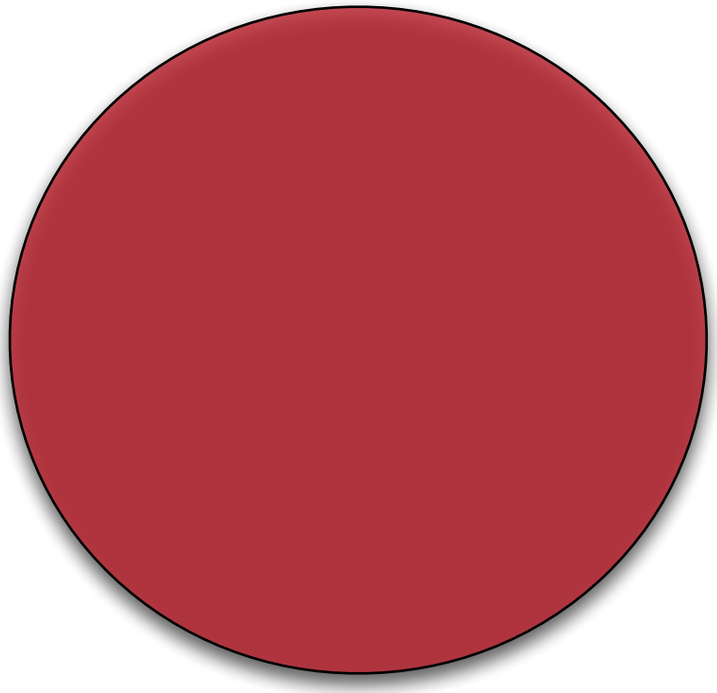
- Effective Praat scripting relies upon mastery of three quite separate domains of knowledge.



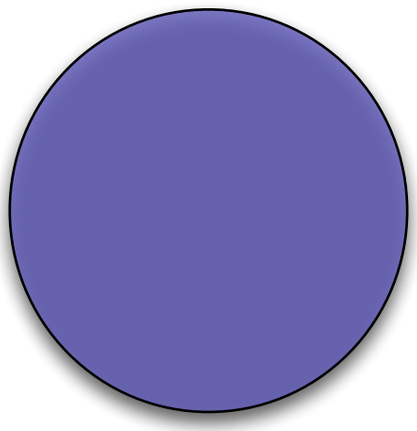


Linguistics domain knowledge

Programming & Debugging

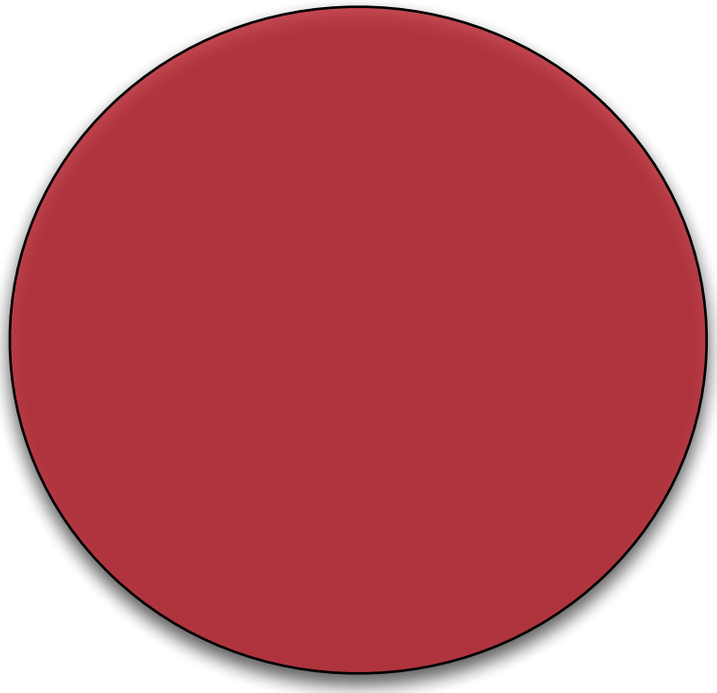


Linguistics domain knowledge

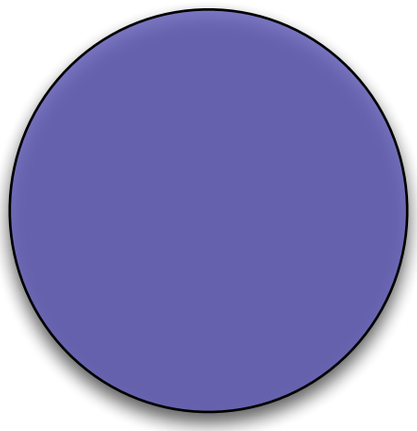


Knowledge of Praat

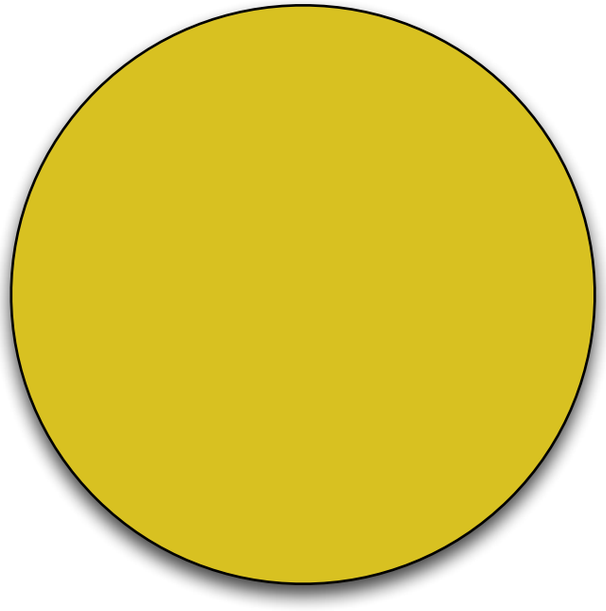
Programming & Debugging



Linguistics domain knowledge



Knowledge of Praat

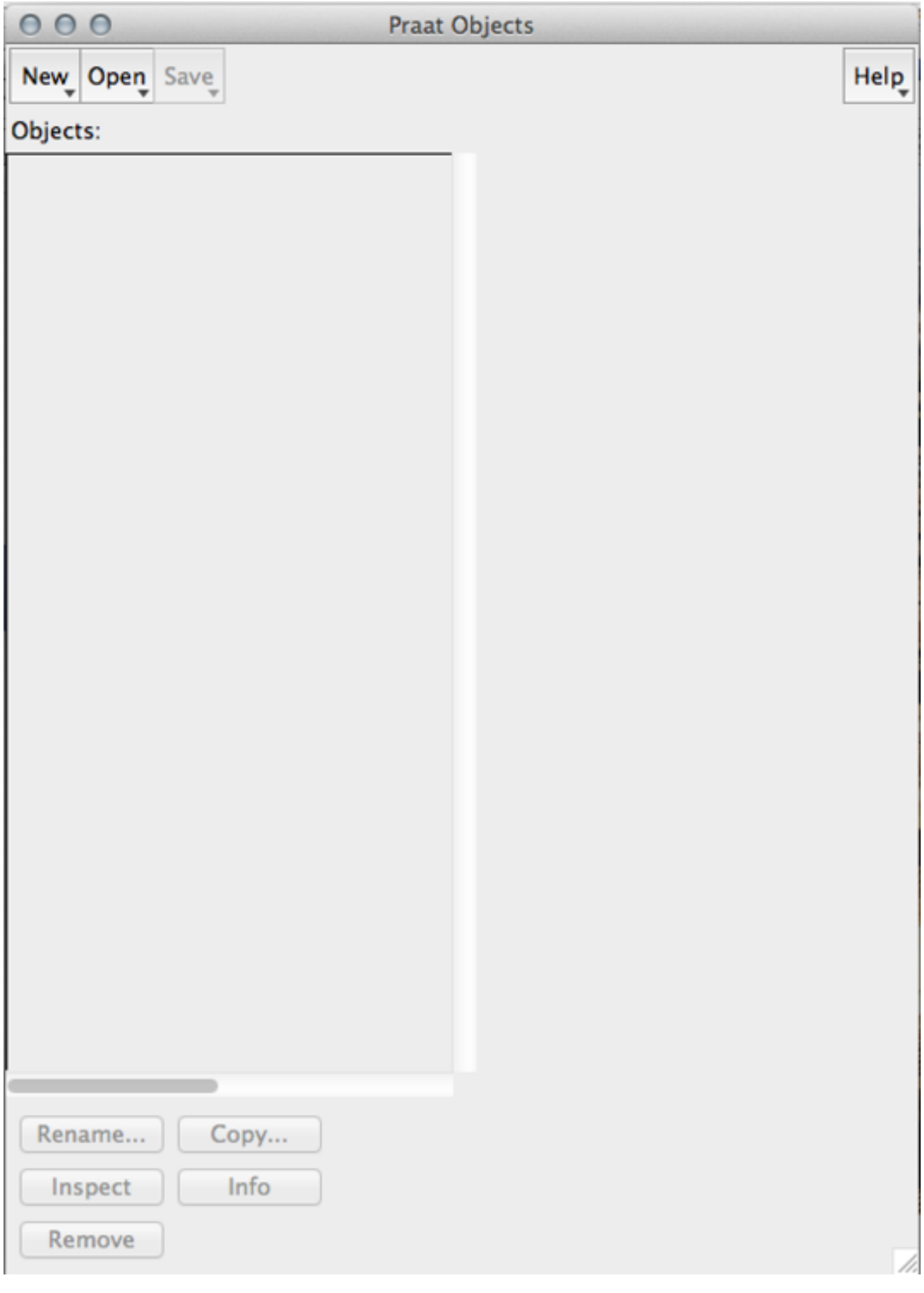


Programming & Debugging

Let’s take a quick tour...

Objects window

- The Objects window is the center of the Praat universe.
- Any data that you load from a file, record with a microphone, extract from another file, or generate with a script will appear as an **object** in this window.
- Each object has a **type** (see available types)
- Each type has a set of associated **data** and **behaviors**



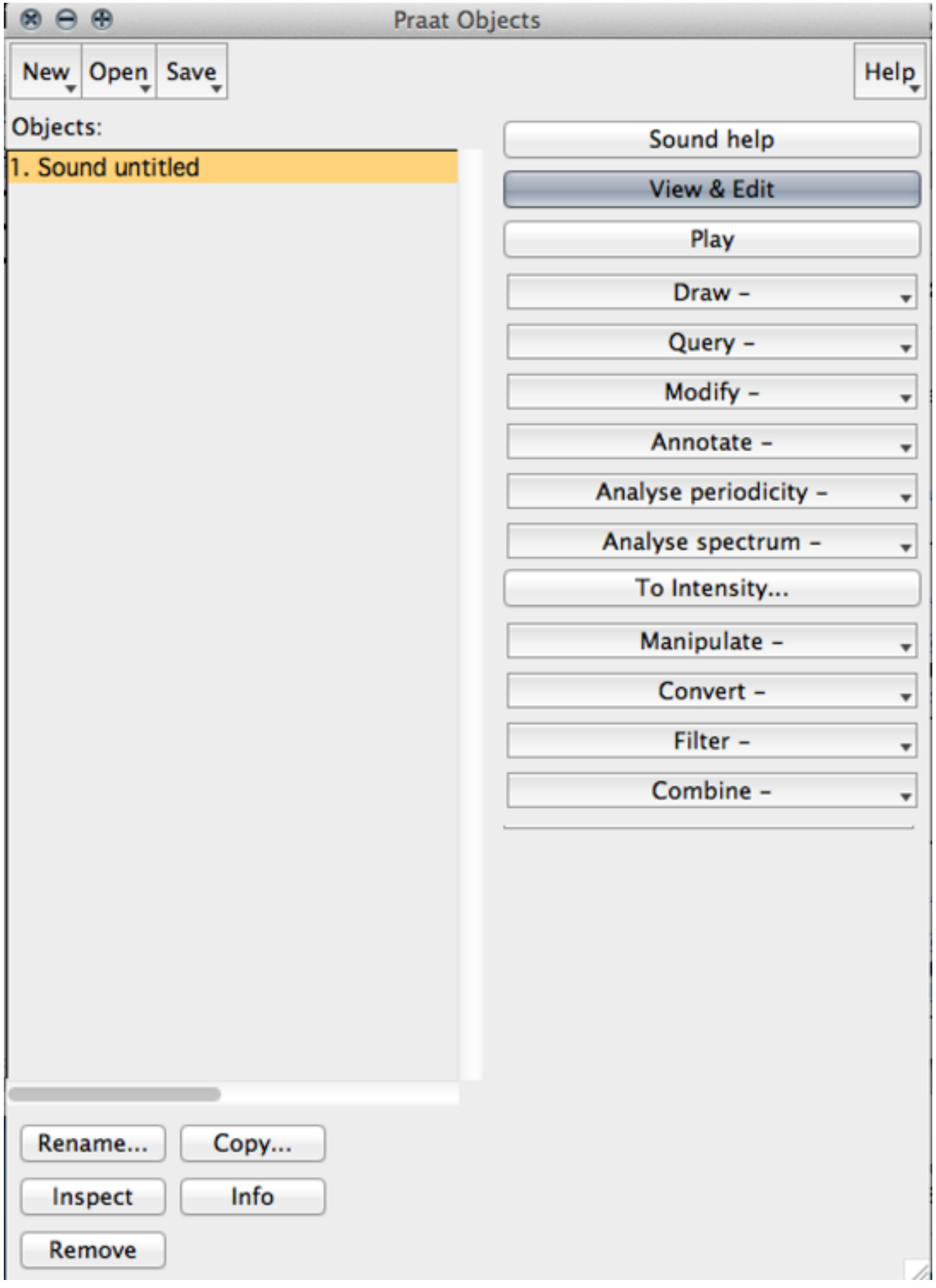
Sound object

- One example of an object is the **sound object**.

- Notice how selecting a sound object populates the right side of the Objects window with **dynamic buttons** and **menus**

- These are specific to the Sound object type.

- At the bottom of the screen are the **fixed buttons**. These never disappear and can't be deleted (more on this later)



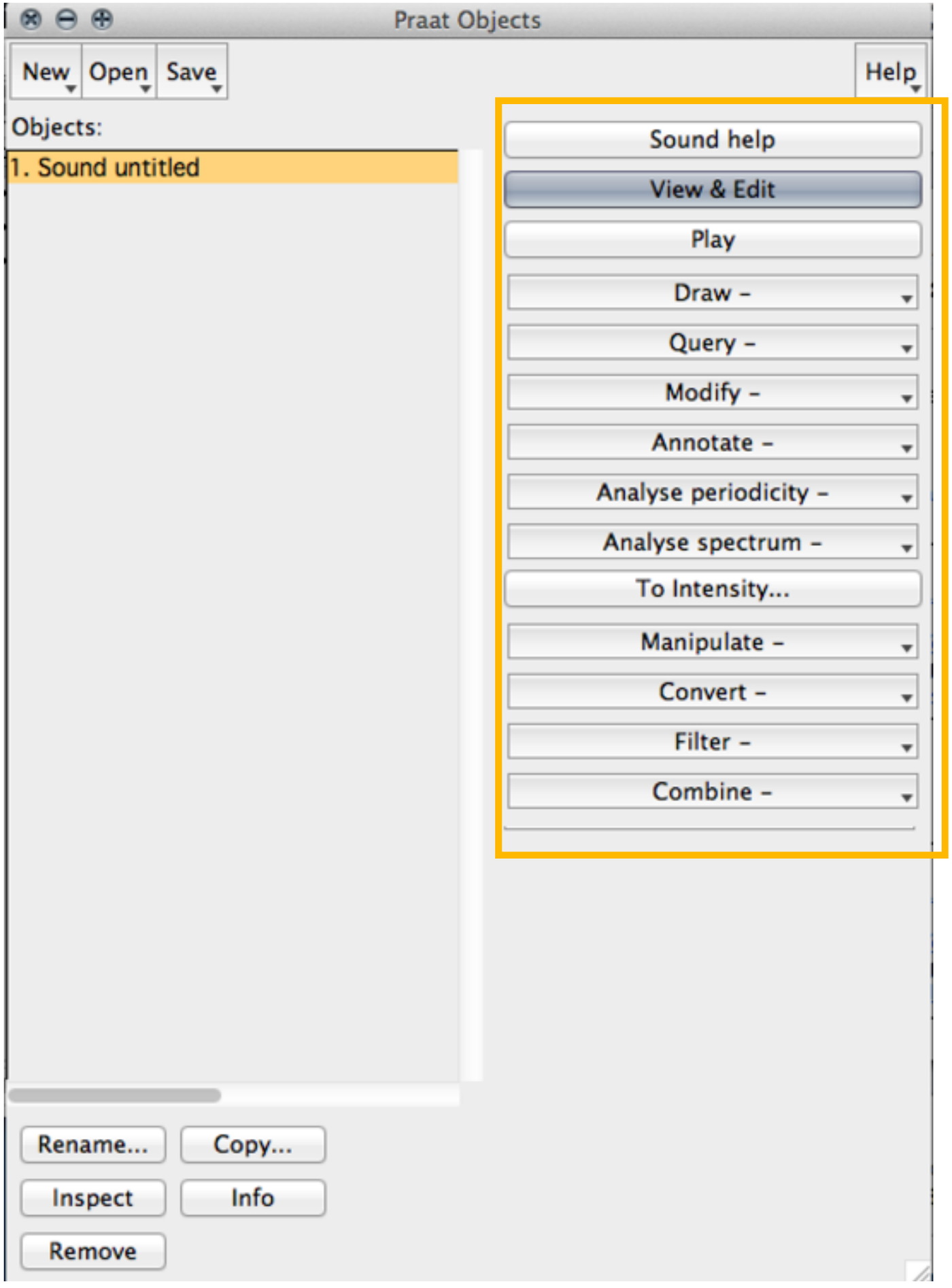
Sound object

- One example of an object is the **sound object**.

- Notice how selecting a sound object populates the right side of the Objects window with **dynamic buttons** and **menus**

- These are specific to the Sound object type.

- At the bottom of the screen are the **fixed buttons**. These never disappear and can't be deleted (more on this later)



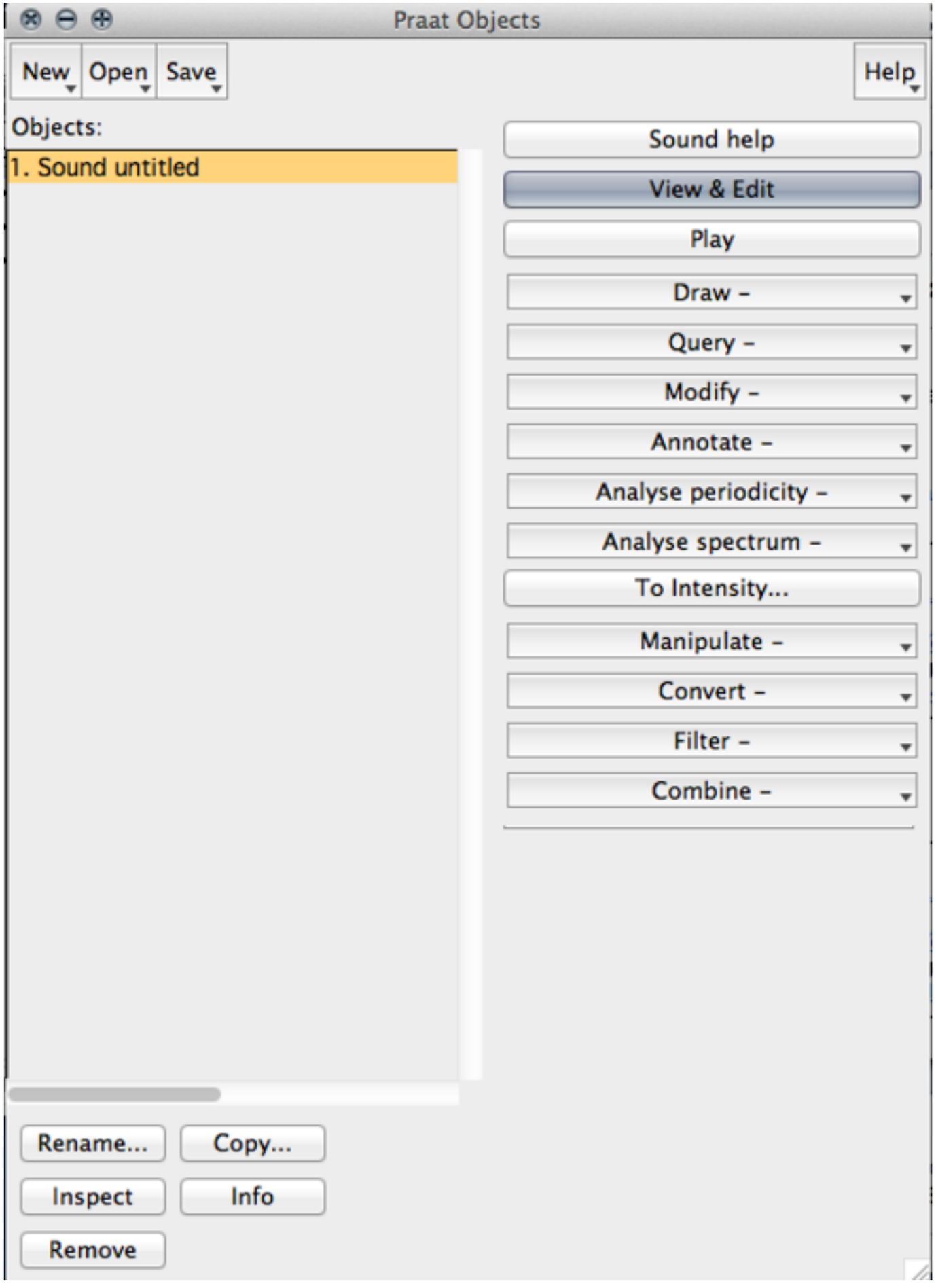
Sound object

- One example of an object is the **sound object**.

- Notice how selecting a sound object populates the right side of the Objects window with **dynamic buttons** and **menus**

- These are specific to the Sound object type.

- At the bottom of the screen are the **fixed buttons**. These never disappear and can't be deleted (more on this later)



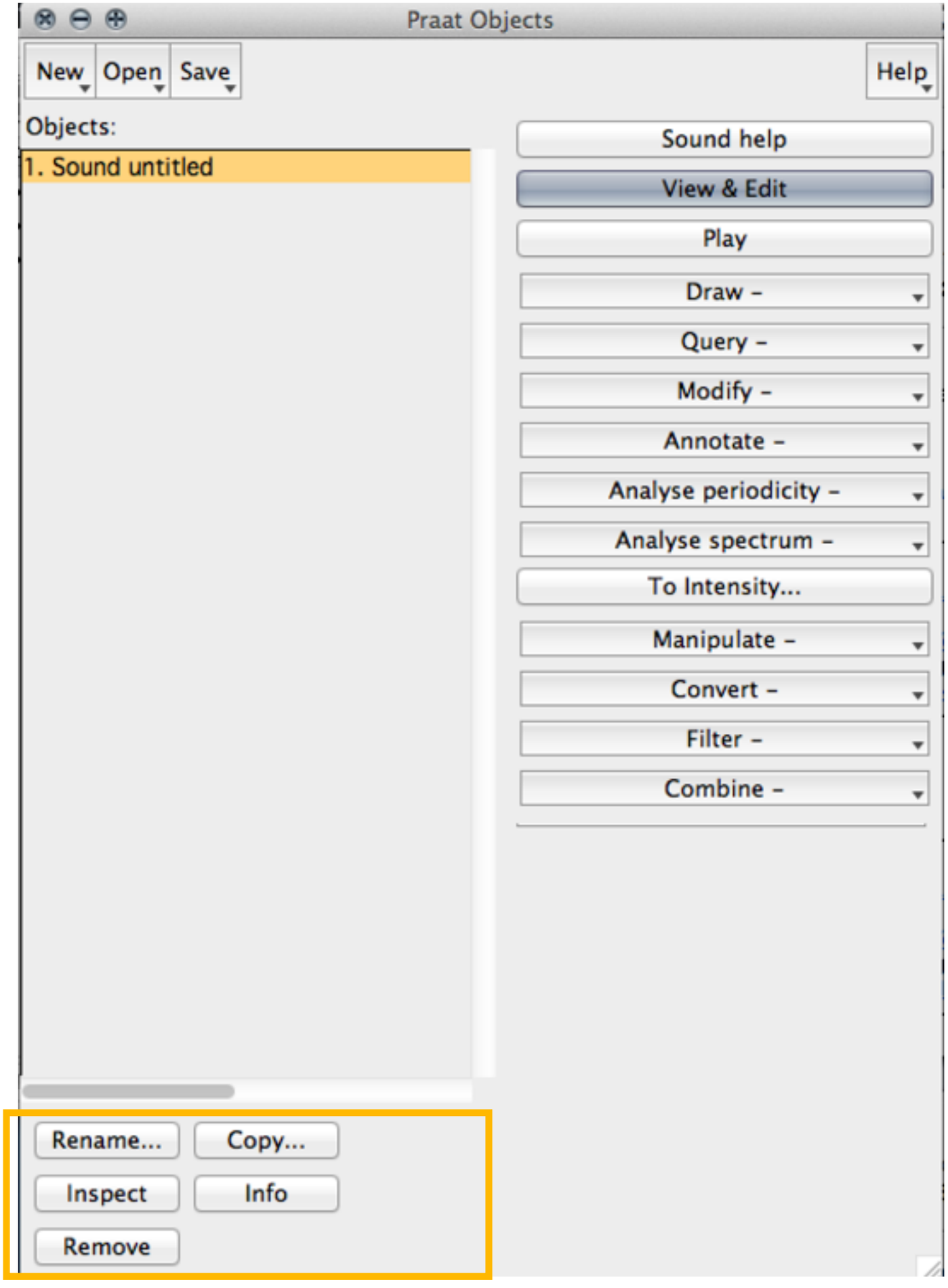
Sound object

- One example of an object is the **sound object**.

- Notice how selecting a sound object populates the right side of the Objects window with **dynamic buttons** and **menus**

- These are specific to the Sound object type.

- At the bottom of the screen are the **fixed buttons**. These never disappear and can't be deleted (more on this later)



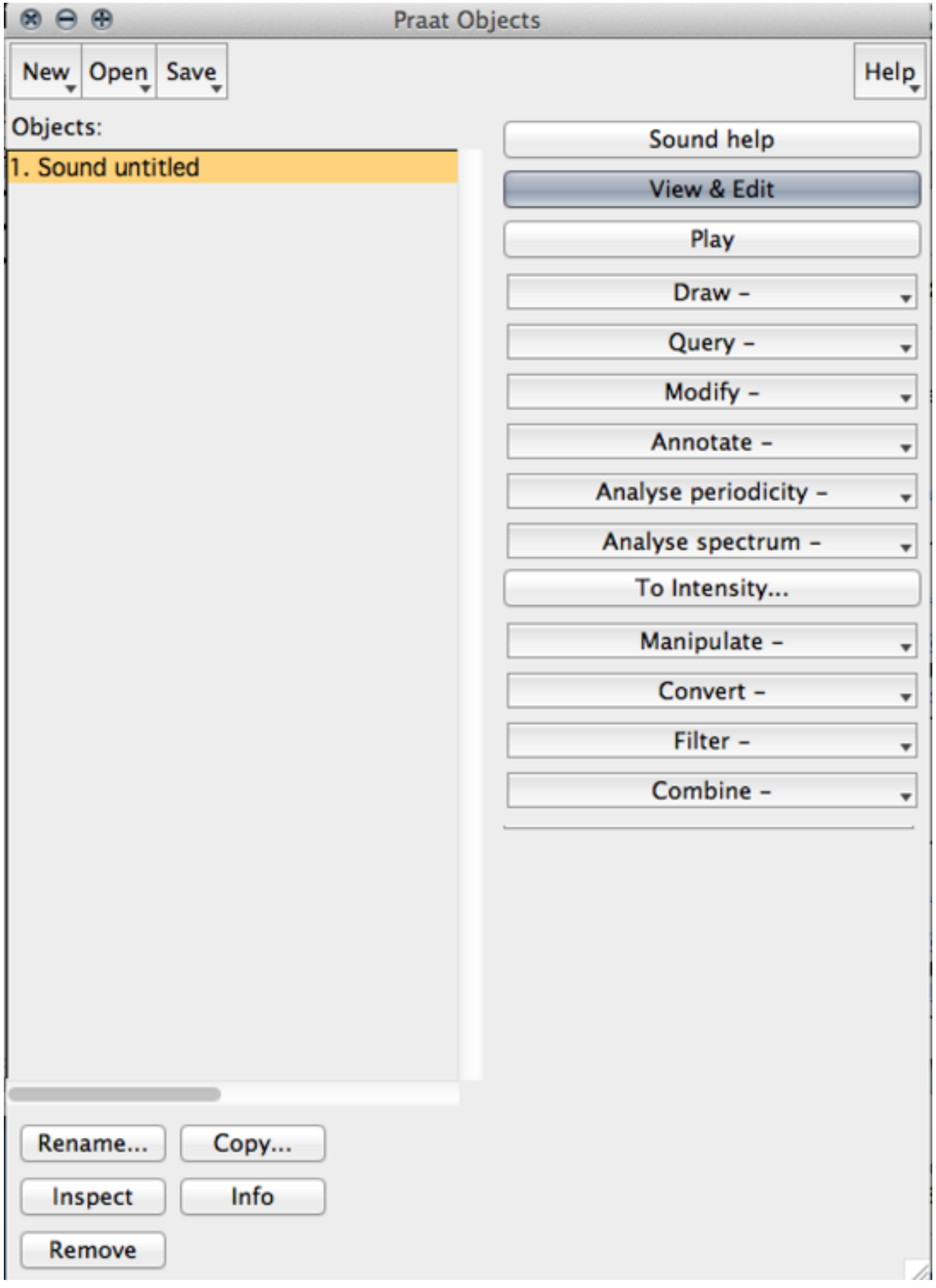
Sound object

- One example of an object is the **sound object**.

- Notice how selecting a sound object populates the right side of the Objects window with **dynamic buttons** and **menus**

- These are specific to the Sound object type.

- At the bottom of the screen are the **fixed buttons**. These never disappear and can't be deleted (more on this later)



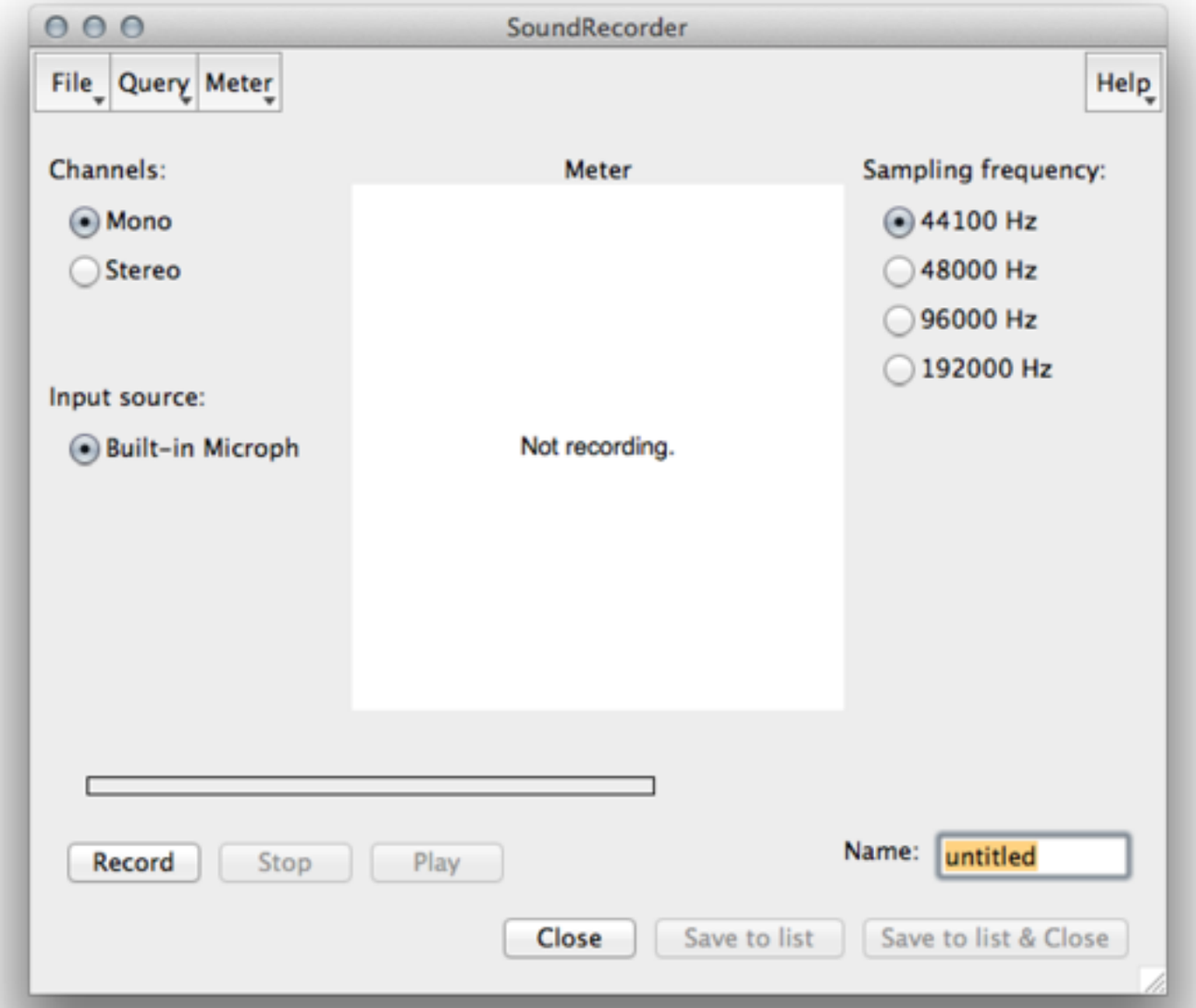
Exercise

- **1.** Open the SoundRecorder window (Objects window → New → Record mono sound (or ⌘R)

- **2.** Record yourself saying something very short (e.g. “doh!”)

- **3.** Save directly to a file (bypassing the Objects window).

- **4.** Now press the button “Save to list & Close”



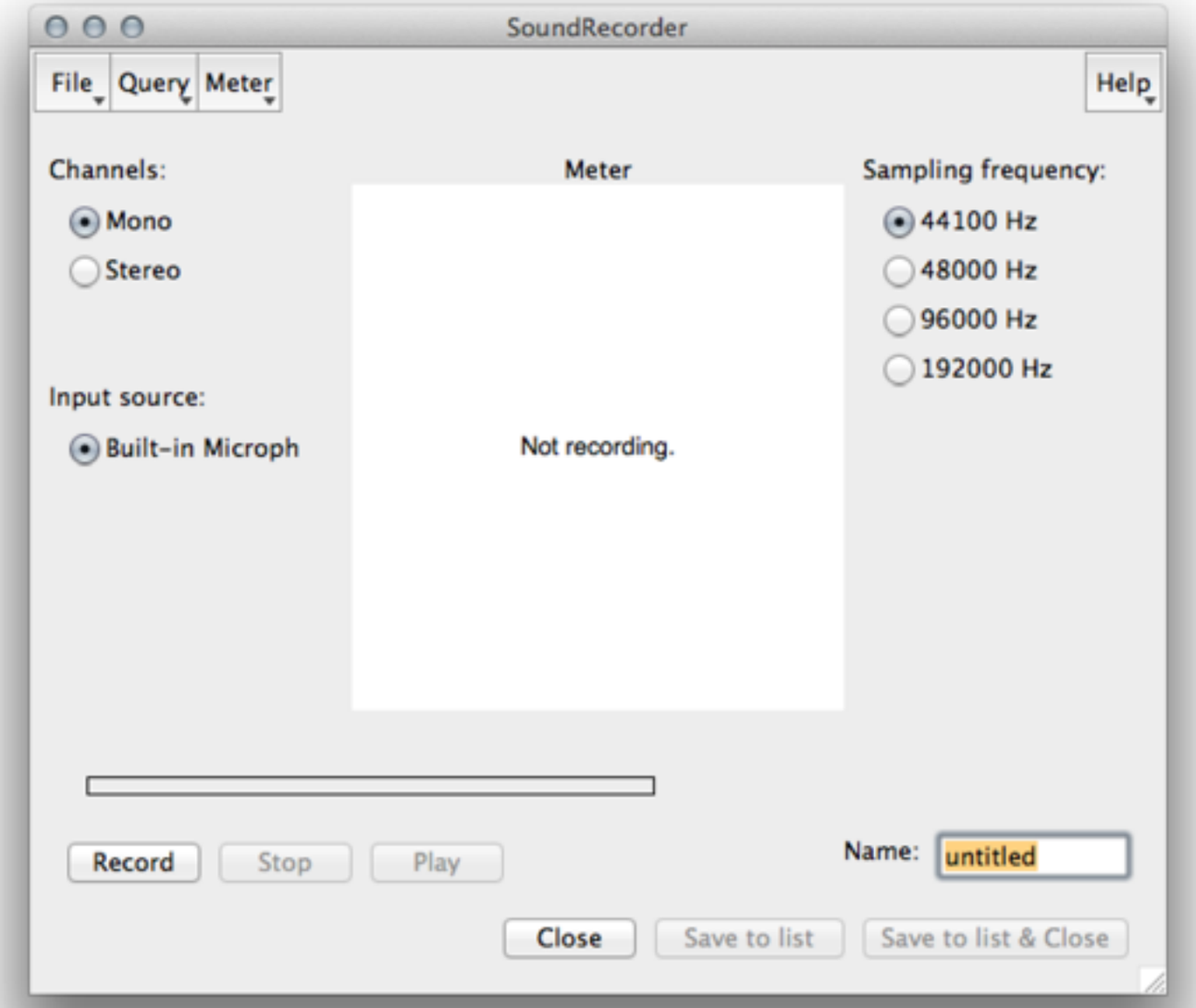
Exercise

- **1.** Open the SoundRecorder window (Objects window → New → Record mono sound (or ⌘R)

- **2.** Record yourself saying something very short (e.g. “doh!”)

- **3.** Save directly to a file (bypassing the Objects window).

- **4.** Now press the button “Save to list & Close”



Q: what buttons are available?

SciptEditor (manual)

- One of the many features Praat offers is a built-in **text editor** that you can use to write and edit your Praat scripts.

- This editor lacks many of the features that are kind of essential for efficient programming (auto-indent, syntax highlighting, brace matching, etc.) but it *does have* three killer features that mean you will end up using it at least occasionally during the development and testing phase of every script you write.

    1. The History Mechanism

    2. Run (⌘R) and Run Selection (⌘T) commands

    3. The ability to add scripts to existing Praat fixed and dynamic menus.

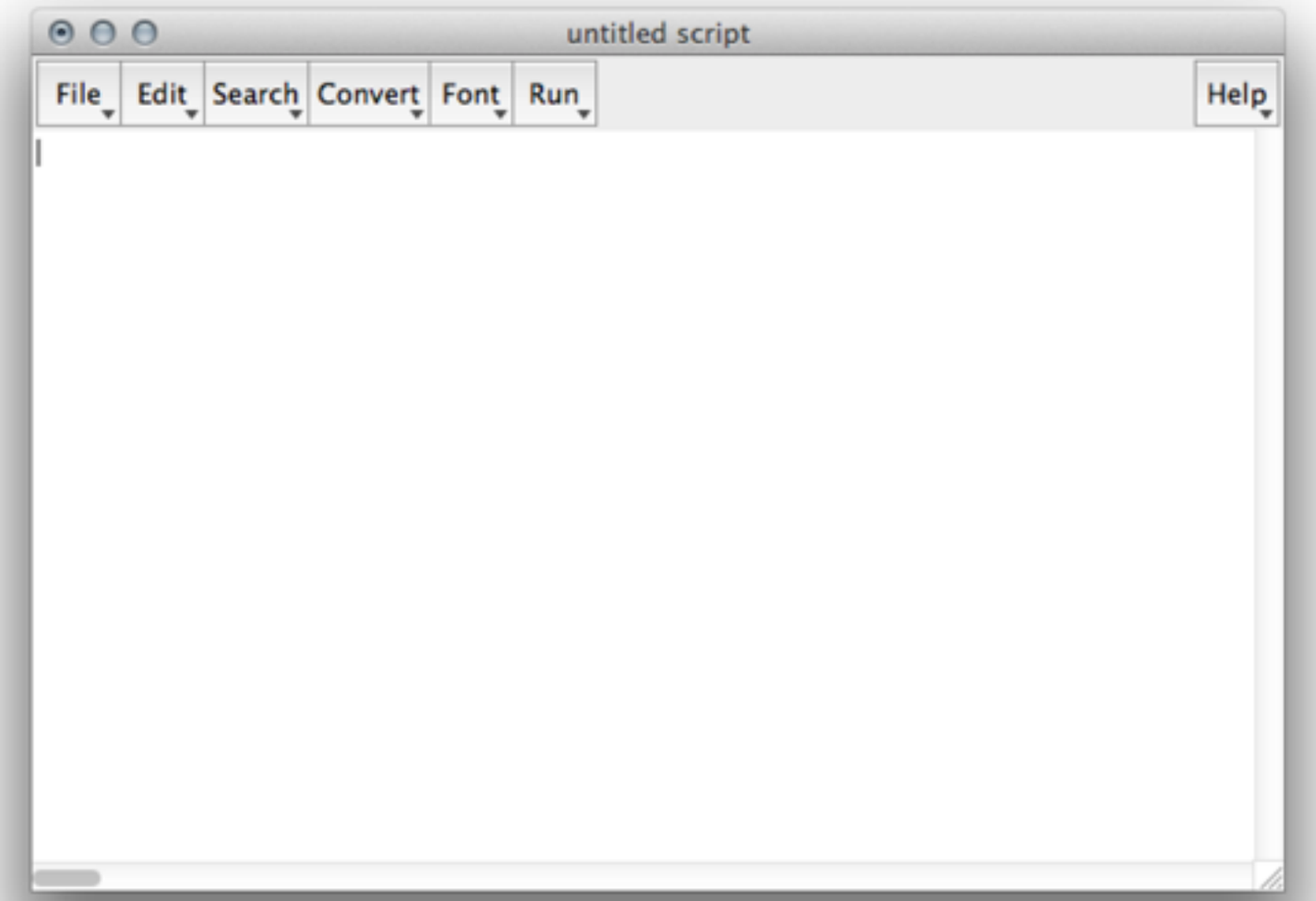
Exercise

1. Open the ScriptEditor window
   (Praat → New Praat script)

2. Press escape [ESC] several times

3. Paste your command history into the ScriptEditor window (Edit → Paste history) or (⌘H)

4. Inspect your new Praat script!



Discussion:

1. What observations can you make about the text that appears?
2. What sorts of commands *do not* appear?

Question

- What would happen if we were to run the Praat script we just created?

- **Answer:**

Question

- What would happen if we were to run the Praat script we just created?

- **Answer:**

What happened when you tried it?

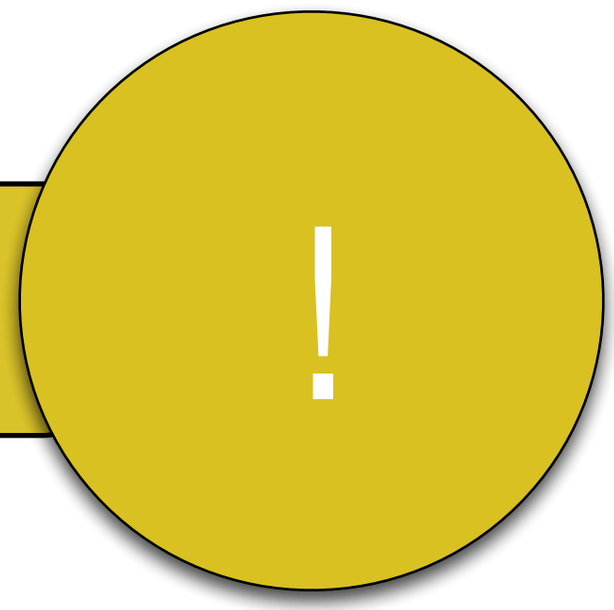
Question

- What would happen if we were to run the Praat script we just created?

- **Answer:**

What happened when you tried it?

Don’t be afraid to experiment!



Aside: Text editor for programming

- Praat scripts (just like python, perl, R, matlab, C, etc...) need to be stored in **plain text files**

- Programs like Word, Pages, LibreOffice Writer, etc. don't do this; their files contain hidden formatting commands and **binary** (as in not text) information

- Praat's built-in ScriptEditor will suffice for short scripts and will be necessary at some point in the life of almost every script

- But you should try out a text editor that is specifically designed for programming and see if you like it

- Learning to use one will save you time in the long run

Aside: Suggested editors

- **Windows**: notepad++ http://notepad-plus-plus.org/download
    - free, open source program, possibly with syntax highlighting
- **Macintosh**: TextWrangler http://www.barebones.com/products/textwrangler/
    - free as in beer (not free as in speech) and also possibly with syntax highlighting
- **Unix (including Mac)**: vim (already installed on your computer)
    - free, insanely powerful, open source program with syntax highlighting
    - steep learning curve but with a game to teach you

Let's write (and break) our first real Praat script

1. Erase any text in the ScriptEditor window (select all and then Edit → Erase)

2. Select the Sound object in the Objects window

3. Clear your command history in the ScriptEditor (Edit → Clear history)

4. Press the dynamic button 'Play'

5. Paste your command history into the ScriptEditor window (Edit → Paste history) or (⌘H)

6. Run your new script (Run → Run) or (⌘R)

7. Create a new TextGrid object (Objects window → Annotate → To TextGrid...) & accept defaults by clicking [ok]

8. Now what happens when you run your script? Why?

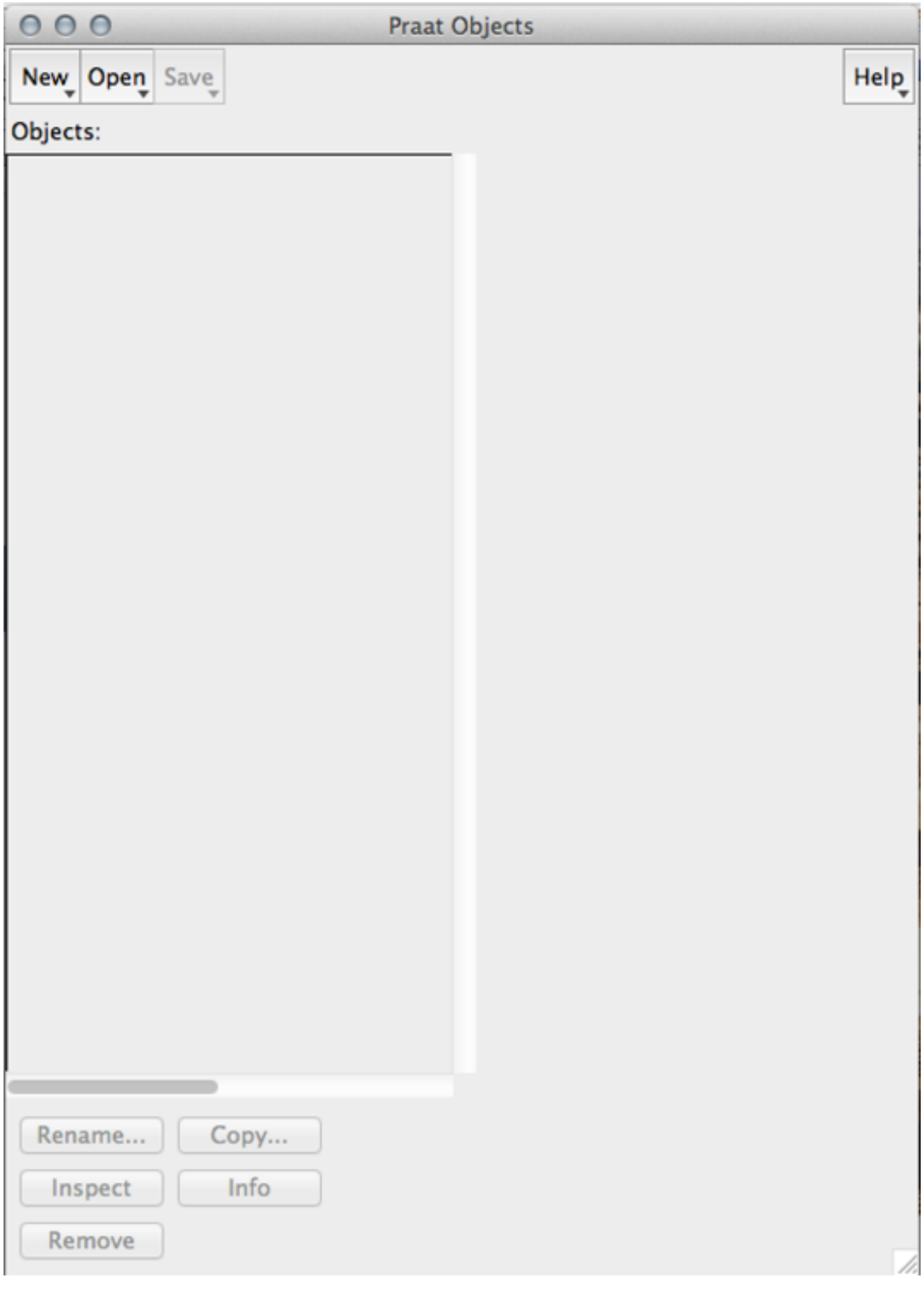
How could we reorder these steps to fix the problem in 8?

Objects window

- Remember: the Objects window is the center of the Praat universe.

- **Before you can use an object, you have to select it!**

- In the GUI (graphical user interface), you do this with a mouse click.



Selecting an object

- Selecting an object makes it active for Praat

- This happens implicitly when you create an object (this is why the new TextGrid broke the play() command –the TextGrid was implicitly selected)

- *Or* you can do it explicitly with selectObject() (manual)

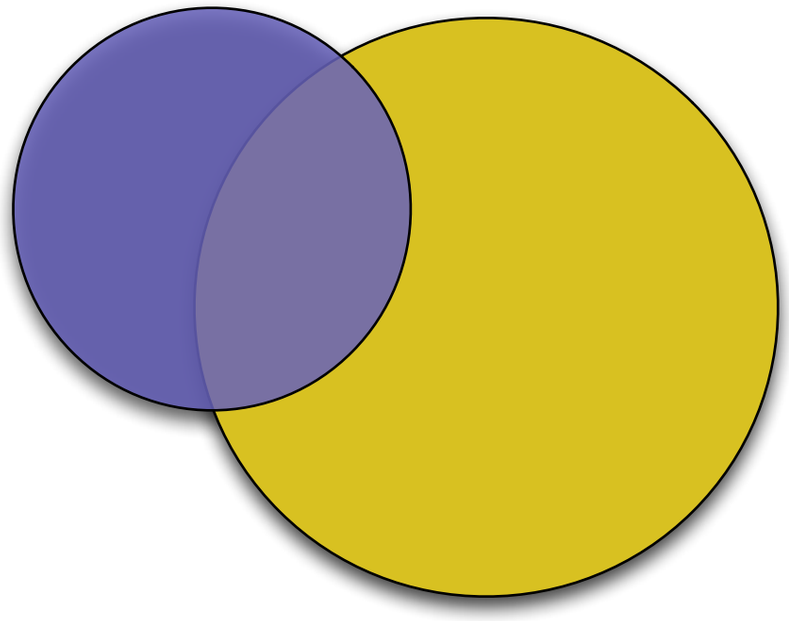
```
selectObject: "Sound untitled"
selectObject: "TextGrid untitled"
selectObject: 1
```

- This is an extremely important tool and will allow you to write much cleaner, safer, and more stable scripts than you are likely to find online!

Praat Scripting

Part 2: Hello, World!

Putting scripts in files...



Files

- Even if you think you won't want to use your script later, you should save it to a text file (because you can crash Praat while scripting).

- By convention, these files end with the extension `.praat`

- I create a 'scripts/' directory associated with each project I am working on and keep all associated scripts there

Comments

```
# lines beginning with a # are comments
; so are lines starting with a semicolon
```

- You can leave a helpful comment like this:

```
    # user needs to hear, and approve, the sound
    Play
```

- Or leave yourself a short **inline comment:**

```
    Play ; user needs to hear, and approve, the sound
```

Best practices for writing comments

- Don't just describe what the code is doing –assume your reader knows the language or can look it up. **Your code should tell readers how you are doing a thing, your comments should tell them why!**

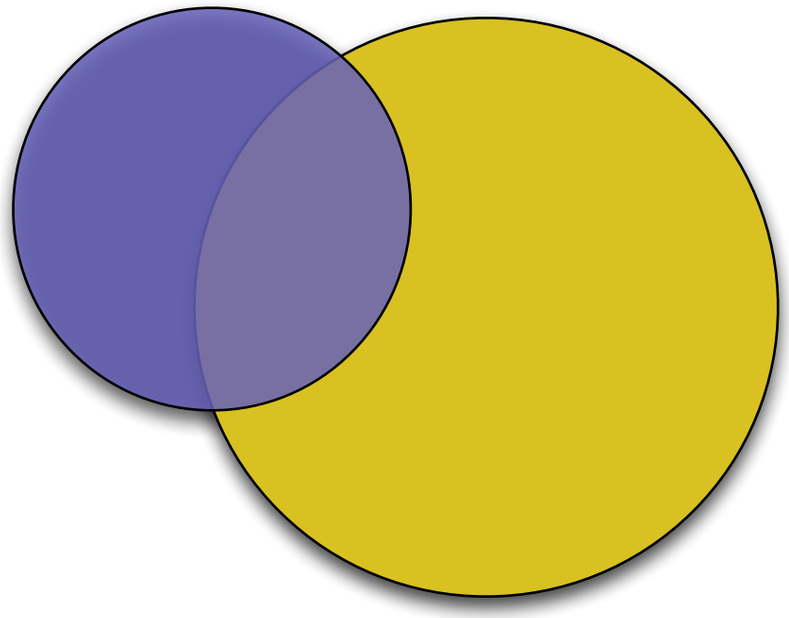
- Avoid comments like:

```
# has various idiosyncrasies
```

- Write comments *while you are coding* to explain why you are doing things (because you will forget)

```
# You can use comments when understanding someone
# else's script. Explain to yourself what is happening!
```

Exercise: Hello, world!



Version 1: The Info window

- Please write the following Praat script and save it as 'hello.praat'

```
# hello, world! in the Info window

writeInfoLine: "hello, world!"
```

```
# another good (short term) use of comments is to leave
# yourself notes while developing or debugging...

# NOTES:
# experiment with changing the text and running this
# script multiple times.
# what EXACTLY does ‘writeInfoLine’ do?
```

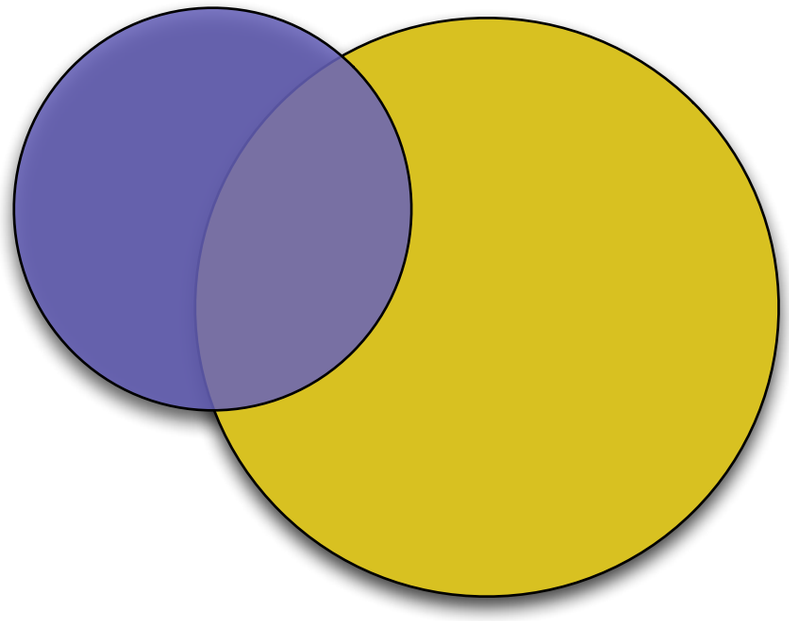
Version 1: The Info window

```
# hello, world! in the Info window

writeInfoLine: "hello, world!"

# experiment with changing the text and running this
# script multiple times.
# what EXACTLY does writeInfoLine do?

# hint: compare it with the behavior of:

appendInfoLine: "it's nice to meet you."



# see http://www.fon.hum.uva.nl/praat/manual/
Scripting_3_1__Hello_world.html
```

Version 2: The Picture window

```
# hello, world! in the Picture window

Erase all
Text top: "yes", "hello, world!"

# question:
# what is the colon in the second line for?
# what does the ‘yes’ in the second line mean?

# see http://www.fon.hum.uva.nl/praat/manual/
Scripting_3_1__Hello_world.html
```

Variables



A more flexible greeting

```
# We can store the greeting in a string variable

greeting$ = "hello, world!"
writeInfoLine: greeting$
```

A more flexible greeting

```
# or get it at run time in a form

form Greet the nice user
    comment Please name the entity we should greet:
    text greeting
endform

writeInfoLine: "hello, ", greeting$, "!"
appendInfoLine: "hello, " + greeting$ + "!"
appendInfoLine: "hello, 'greeting$'!"

# question: what do the plus signs do in the second line?
# question: what are the single quotes for in the third line?
```

Variables

- As we've just seen, a **variable** works like a named bucket that can store an arbitrary value and return it when you call for the bucket by name again

- A variable is an address in your computer's memory. This address points at a **value**.

- The variable is on the left side of the equals operator (=) and the value of that variable is on the right side

```
# in Praat, a variable assignment looks like this:
greeting$ = "world!"
```

- We can read this as: the string variable 'greeting' points at the string constant "world!"

Variable names

- Variable names in Praat must begin with a lowercase ascii letter (a through z)
- The following variable names will generate syntax errors:
  - 1variable (but variable1 or var1abl3 are fine)
  - Variable (but variable is fine)
  - üvariable (but värïäblë is fine)
  - åvariable (but vårïable is fine)

Uppercase & lowercase in Praat

- Not only are all *variables* lowercase-initial in Praat, but all script-specific commands are also lowercase-inititial.
    - Examples include: `demo, do, form, writeInfoLine, if, while,` selectObject, etc.
- All of Praat's Button commands (i.e. menu commands, Editor commands, dynamic and fixed commands for objects) begin with an uppercase letter
    - Examples include: `Play, Paint..., About Praat..., Select..., To Spectrogram...`, etc.

Variable names: available characters

- In general, it is safe to use letters, numbers, and underscore (_) in variable names.

- Praat variable names cannot contain punctuation

- Non-ascii characters can be used as data or constants in Praat scripts, but not as variable names

```
# legal praat code
caution$ = "حذر"
writeInfoLine( caution$ )

# syntax error
慎重$ = "caution"
writeInfoLine( 慎重$ )
```

Basic variable types

- Praat has two basic variable types:
    - **string variables**, like we have used so far, end in a $ character and can hold anything from the empty string to (at least) four copies of War and Peace.
    - **numeric variables** can be integers or real (decimal) numbers
- As with object types, variable types dictate what actions and behaviors are available for a variable.

Numeric variables

- Praat's numeric variables can hold values between -1,000,000,000,000,000 and +1,000,000,000,000,000 or real numbers between -10^{308} and 10^{308}

- Numeric constants may not contain commas (try it, see what happens)

- Decimal numbers must have at least a zero to the left of the decimal point.

- Numeric assignments look like these:

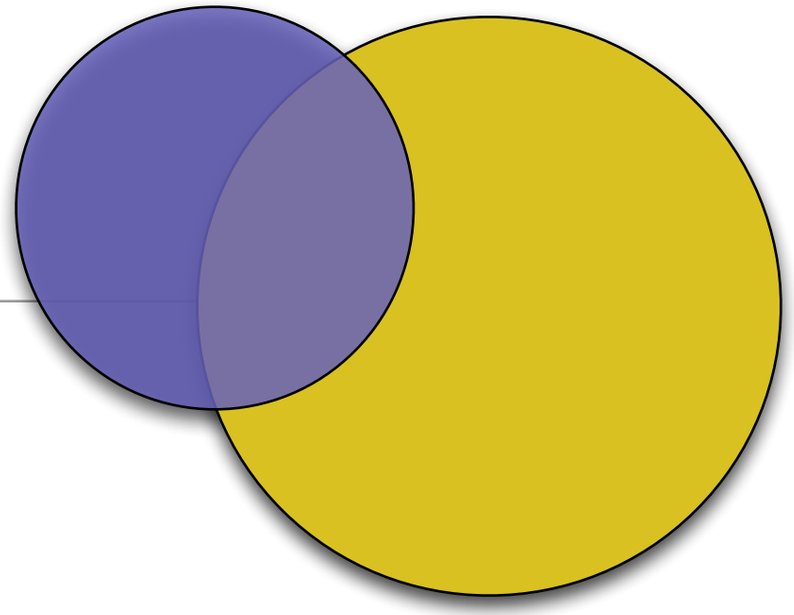
```
c = 35000       ; cm/s speed of sound in air
length = 17.5   ; cm vocal tract
f1_ə = (2 * 1 - 1) * c /  ( 4 * length )
f2_ə = (2 * 2 - 1) * c / ( 4 * length )

writeInfoLine( "According to the tube theory, F1 is ", f1_ə, " and F2 is
", f2_ə )
```

Exercise

1. Create a string variable whose value is a number.
   (this may take a few tries)

2. Try multiplying this string variable by 2

Exercise



1. Create a string variable whose value is a number.
   (this may take a few tries)

2. Try multiplying this string variable by 2

- Hint: You can convert a string to a number or a number to a string.

```
a$ = string$: a

a = number: a$
```

Extended Exercise 1

- Record yourself saying “Hello, world!” in any language that you like and save to a wav file.

- Write a Praat script to read this wav file and paint a spectrogram of this sound in the Picture window (please show 0 to 8000Hz). Let Praat “garnish” this spectrogram.

- Your script should also add a title to the figure of the form: “Dia duit, domhan!” Irish spoken by Caoimhín Mac an Gobhann

- Begin the script with a comment that gives your name and a brief description of what the script is meant to do.

- E-mail me the script file (YourName.praat), your wav file, and a PDF (or EPS on Windows/Linux) of the spectrogram if you want me to see it!

Extended Exercise 1: Review

EA1: Example solution

```
sound = Read from file: "/Users/clunis/Desktop/untitled.wav"
sgram = To Spectrogram: 0.005, 8000, 0.002, 20, "Gaussian"

Paint: 0, 0, 0, 8000, 100, "yes", 50, 6, 0, "yes"
Text top: "no", """Hello, world!"" in Irish spoken by Kevin McGowan"

Save as PDF file: "DiaDuitDomhan.pdf"
```

EA1: Example solution (better)

```
# Paint a publishable spectrogram
#
# Reads a sound from a wav file, creates and paints a spectrogram of that
# sound, adds a meaningful title, and saves a PDF file.
#
# 6-25-2013 Kevin B. McGowan <kbmcgowan@stanford.edu>

sound = Read from file: "/Users/clunis/Desktop/untitled.wav"
sgram = To Spectrogram: 0.005, 8000, 0.002, 20, "Gaussian"

Paint: 0, 0, 0, 8000, 100, "yes", 50, 6, 0, "yes"
Text top: "no", """Hello, world!"" in Irish spoken by Kevin McGowan"

Save as PDF file: "DiaDuitDomhan.pdf"

removeObject( sound )
removeObject( sgram )
```

EA1: Example solution (best)

```
# Paint a publishable spectrogram
#
# Reads a sound from a wav file, creates and paints a spectrogram of that
# sound, adds a meaningful title, and saves a PDF file.
#
# 6-25-2013 Kevin B. McGowan <kbmcgowan@stanford.edu>

sound = Read from file: "/Users/clunis/Desktop/untitled.wav"
sgram = To Spectrogram: 0.005, 8000, 0.002, 20, "Gaussian"

Erase all
Select outer viewport: 0, 6, 0, 4
Paint: 0, 0, 0, 8000, 100, "yes", 50, 6, 0, "yes"
Text top: "no", """Hello, world!"" in Irish spoken by Kevin McGowan"

Save as PDF file: "DiaDuitDomhan.pdf"

removeObject: sound
```

EA1: Example solution (more bester)

```
#
# 6-25-2013 Kevin B. McGowan <kbmcgowan@stanford.edu>

sound = Read from file: "/Users/clunis/Desktop/untitled.wav"
sgram = To Spectrogram: 0.005, 8000, 0.002, 20, "Gaussian"

Erase all
Select outer viewport: 0, 6, 0, 4
Paint: 0, 0, 0, 8000, 100, "yes", 50, 6, 0, "yes"
Text top: "no", """Hello, world!"" in Irish spoken by Kevin McGowan"

if macintosh
    Save as PDF file: "DiaDuitDomhan.pdf"
else
    Save as EPS file: "DiaDuitDomhan.eps"
endif

removeObject: sound
removeObject( sgram )
```

Quotes

- double quotes ("") are used to define a string literal
  e.g. `greeting$ = "world!"`

- Single quotes can be used to force Praat to substitute a string variable for its value. e.g. `appendInfoLine: "hello, 'greeting$'!"`

Quotes

- double quotes ("") are used to define a string literal
  e.g. `greeting$ = "world!"`

- Single quotes can be used to force Praat to substitute a string variable for its value. e.g. `appendInfoLine: "hello, 'greeting$'!"`

This use of single quotes for variable substitution used to be (prior to 2012) very common in Praat and resulted in ugly, difficult syntax and strange bugs. It should be avoided now. (see Boersma's message on the subject)

More on quotes

- Single quotes cannot be used to define a string constant. The following generates a syntax error:

- To embed double quotes in a double quoted string, double the double quotes:

```
greeting$ = 'world!'
```

- To embed double quotes in a double quoted string, double the double quotes:

```
writeInfoLine: "And he was all, ""whoa."" and I"
appendInfoLine: "was all, ""right?"""
```

More on quotes

- Single quotes cannot be used to define a string constant. The following generates a syntax error:

- To embed double quotes in a double quoted string, double the double quotes:

```
*greeting$ = 'world!'
```

- To embed double quotes in a double quoted string, double the double quotes:

```
writeInfoLine: "And he was all, ""whoa."" and I"
appendInfoLine: "was all, ""right?"""
```

Reference: Numeric Operators & Precedence (manual)

precedence -->

negation (-) and exponentiation (^)	2^2^3 = 2^8 = 256 5^-3 = 5^{-3} 0.008	(right to left)
multiplication (*) and division (/)	1 / 4 * 2 = 0.5 1 / (4 * 2) = 0.125	
addition (+) and subtraction (-)	7 * 2 + 2 = 16 7 * -2 -2 = -16	
comparison operators	eq. (=), not eq. (<>), lt (<), gt (>), lt or eq. (<=) and gt or eq. (>=)	
not (!), and (&), or (|)	not 7 + 3 = 10 6 & 6 x > 5 and x < 10	

<-- precedence

Reference: String Comparison (manual)

a$ = b$
true (1) if the strings are equal, and false (0) otherwise.

a$ <> b$
1 if the strings are unequal, and 0 otherwise.

a$ < b$
1 if a$ precedes b$ in ASCII sorting order.

a$ > b$
1 if a$ comes after b$ in ASCII sorting order.

a$ <= b$
1 if a$ precedes b$ in ASCII sorting order, or if the strings are equal.

a$ >= b$
1 if a$ comes after b$ or the two are equal.

Praat Scripting

Part 3: Variables and Conditionals

Making decisions

- Sometimes it is necessary for a program to be capable of doing different things depending on input or calculations

- As in natural language, computer science calls the expressions capable of introducing multiple possible worlds **conditionals**

- The simplest form of a conditional has the following structure:

```
# note, this is not in any particular programming
# language.  This is an example of pseudocode.
if <test>
     what to do if the test is true
endif
```

Making decisions

- Notice that this pseudocode does nothing at all if the test is false.

- Here's an example with real Praat code:

```
# what does this code do when executed?
if 7 > 9
     writeInfoLine: "hello, world!"
endif
```

if, elsif, else, endif

- Praat has four reserved words set aside for conditional expressions:
- **if** – introduces a conditional, must include a test.
- **elsif** – allows for the inclusion of another possible true outcome (note the lack of 'e'! Can also be spelled 'elif'.
- **else** – if none of the preceding tests was true, execute this **block** of code
- **endif** – ends the conditional

Space best practices

- Notice the indentation:

```
if 7 >= 9
     # we have dropped through a wormhole into
     # a parallel (and inconsistent) universe
     writeInfoLine( "hello, not our world!" )
else
     # the physical laws of our universe apply
     writeInfoLine( "hello, world!" )
endif
```

- I use 4 spaces to indent lines inside a block (some people prefer 2)

- Avoid tabs, there is no standard on the placement and display of tab stops

Conditional example

```
num = -3

if num = 0
     writeInfoLine: "number *is* 0."
elsif num < 0
     writeInfoLine: "number is less than 0."
else
     writeInfoLine: "number is greater than 0."
endif

appendInfoLine: ”wasn't that fun?"
```

Exercise: Random Test

- Praat has a function `randomInteger(min, max)` which is essentially telling Praat to pick a number between min and max (inclusive). e.g. `randomInteger(1, 10)` could be read "pick a number from 1 to 10".

- Please write a script to:

1. Generate a random number between 1 and 10 and store it in a numeric variable.

2. Test whether this number is greater than or equal to 6. If so, write "The number was 6 or higher." to the Info window.

3. If the number is not equal to or greater than 6, write "Your number was 5 or lower." to the Info window.

Exercise: Guess my number

- **[Difficult]** Now that you have a working random number script, modify it to:

1. Prompt the user with a form that says: "I am thinking of a number between 1 and 10. Can you guess it?"

2. Use the form to get an `integer` called `guess` from the user

3. Tell the user if their `guess` was (1) correct (2) too high or (3) too low

String Concatenation and Truncation

a$ + b$
Concatenates the two strings.

a$ - b$
Subtracts the second string from the end of the first.

```
soundFile$ = "bananas.wav"
soundFile$ = soundFile$ - ".wav"
tgFile$ = soundFile$ + ".TextGrid"
# tgFile$ now points to "bananas.TextGrid"
```

But notice!

```
soundFile$ = "bananas.aiff"
soundFile$ = soundFile$ - ".wav"
tgFile$ = soundFile$ + ".TextGrid"
# tgFile$ now points to "bananas.aiff.TextGrid"
# to fix this you want string functions
```

Using functions

- The behaviors available to perform on the different variable types are called **functions**

- We have already seen two examples of these in the `string$:` and `number:` functions

- Praat has a length() function that simply calculates and returns the length of a single string **argument**

```
string$ = "what has it got in its pocketses?"
length = length: string$
writeInfoLine: "The string '", string$, "' is ", length,
..." characters long."
```

- (you can break up a long line by inserting ellipses after newline)

Testing functions

- Remember that, in a conditional, `0` is false and any other number is true.

- You can verify this yourself with something like:

```
n = -4
if n
     writeInfoLine: "beauty!"
endif
```

- We can take advantage of this fact to use functions in conditionals...

Testing functions (cont)

- For example, to test if a file is a wav file or a TextGrid, you could write:

```
if endsWith: filename$, ".wav"
     # file is a wav, do sound file things to it
     # ...
elsif endsWith: filename$, ".TextGrid"
     # file is a TextGrid, do TextGrid file things to it
     # ...
else
     # file is neither (or the case didn't match )
     # ...
endif
```

Reference: Available functions

- The Praat documentation lists the available functions:
  - Mathematical functions (manual: Formulas 4)
  - String functions (manual: Formulas 5)

Predefined variables

- Praat comes with a number of pre-defined variables and constants, some of them are:

- **Numeric**: `praatVersion` stores the current version of Praat you are running. `macintosh`, `windows`, and `unix` are true (1) if you are on that platform. `pi`, `e`, and `undefined` (`manual: constants`)

- **String**: `praatVersion$` version of Praat as a string. `newline$`, `tab$` store your platform's take on newline and tab. `defaultDirectory$` is the path containing the script file. `homeDirectory$`, `preferencesDirectory$`, and `temporaryDirectory$` provide useful directory paths.

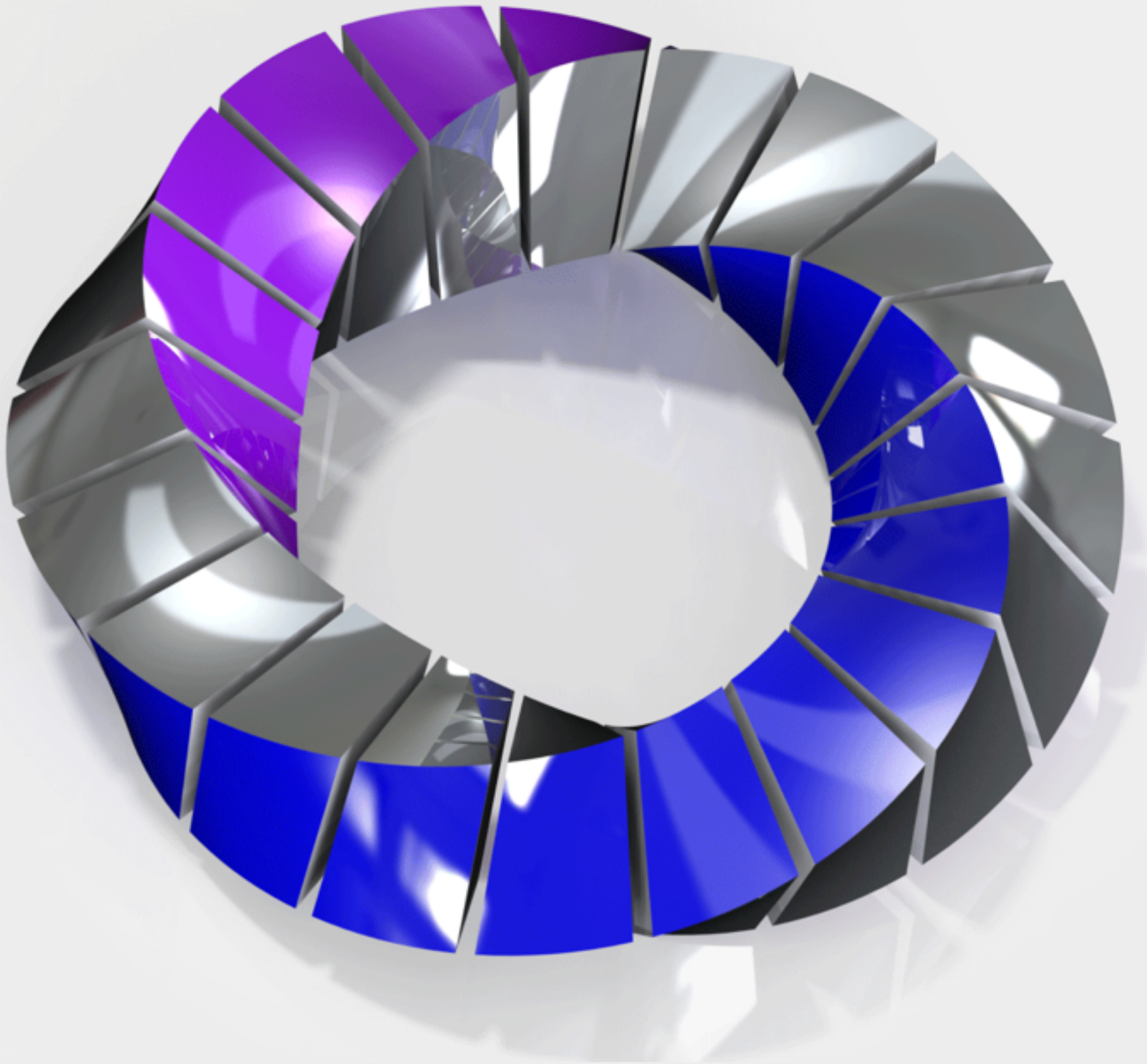
Exercise: Detect O/S

- What O/S is your script running on?

- Write a script that outputs one message if it is running on Macintosh, another if Windows, and another message if it is running on Unix.

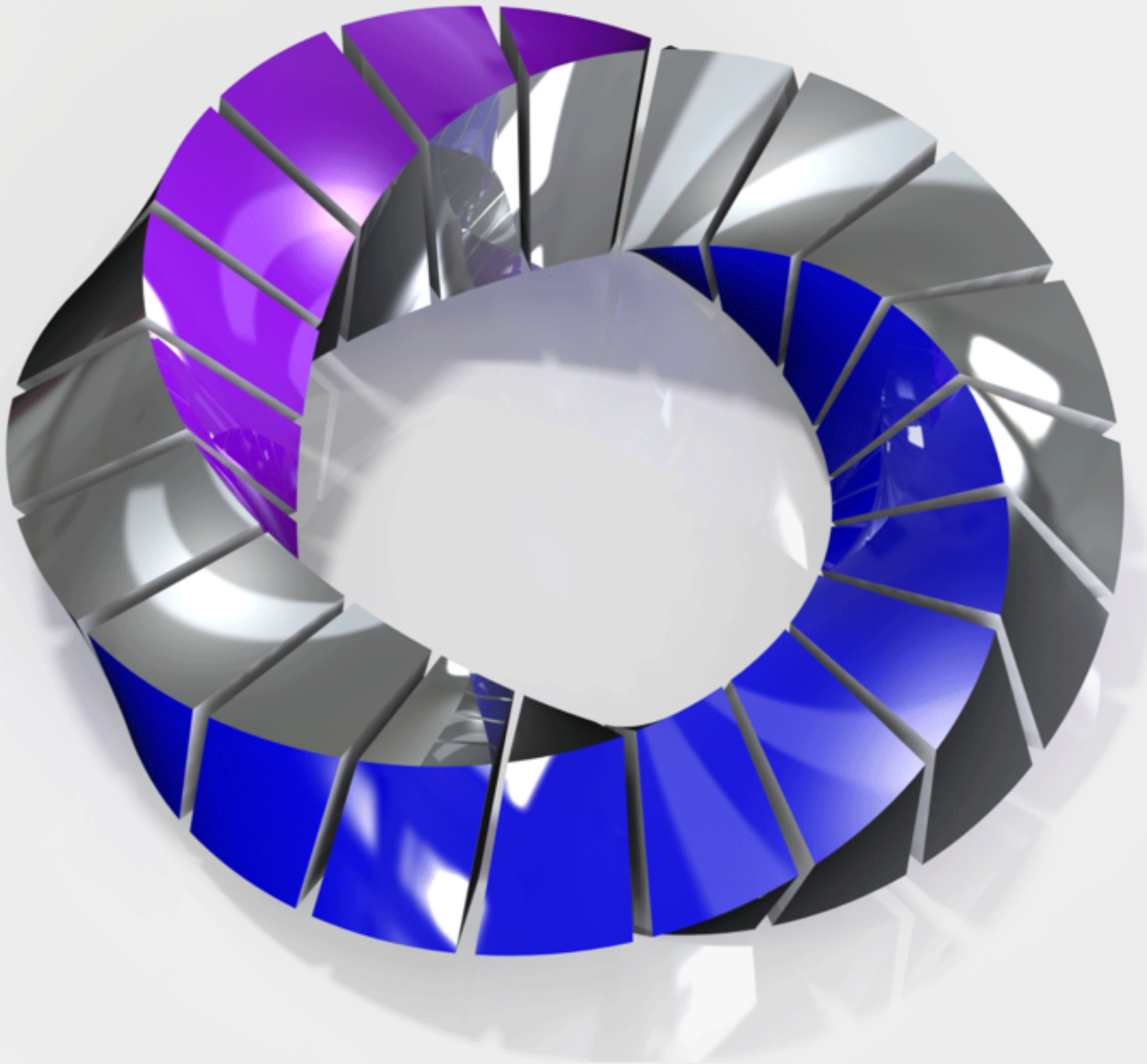
- Include the version of Praat in your output.

- Challenge: try to only include the **writeInfoLine**: (or **appendInfoLine**) one time.



Praat Scripting

Section 2: Textgrids & Loops

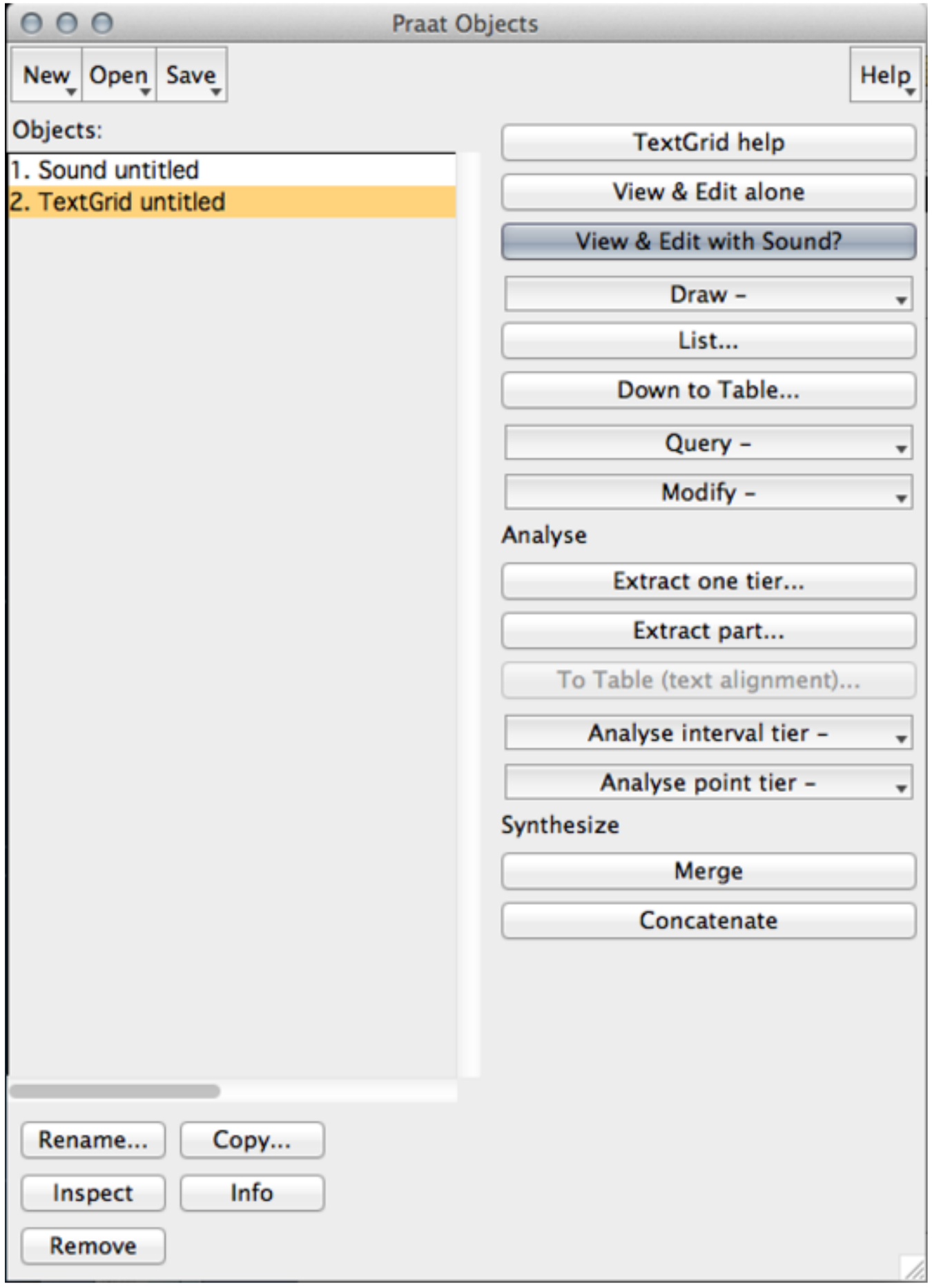


Praat Scripting

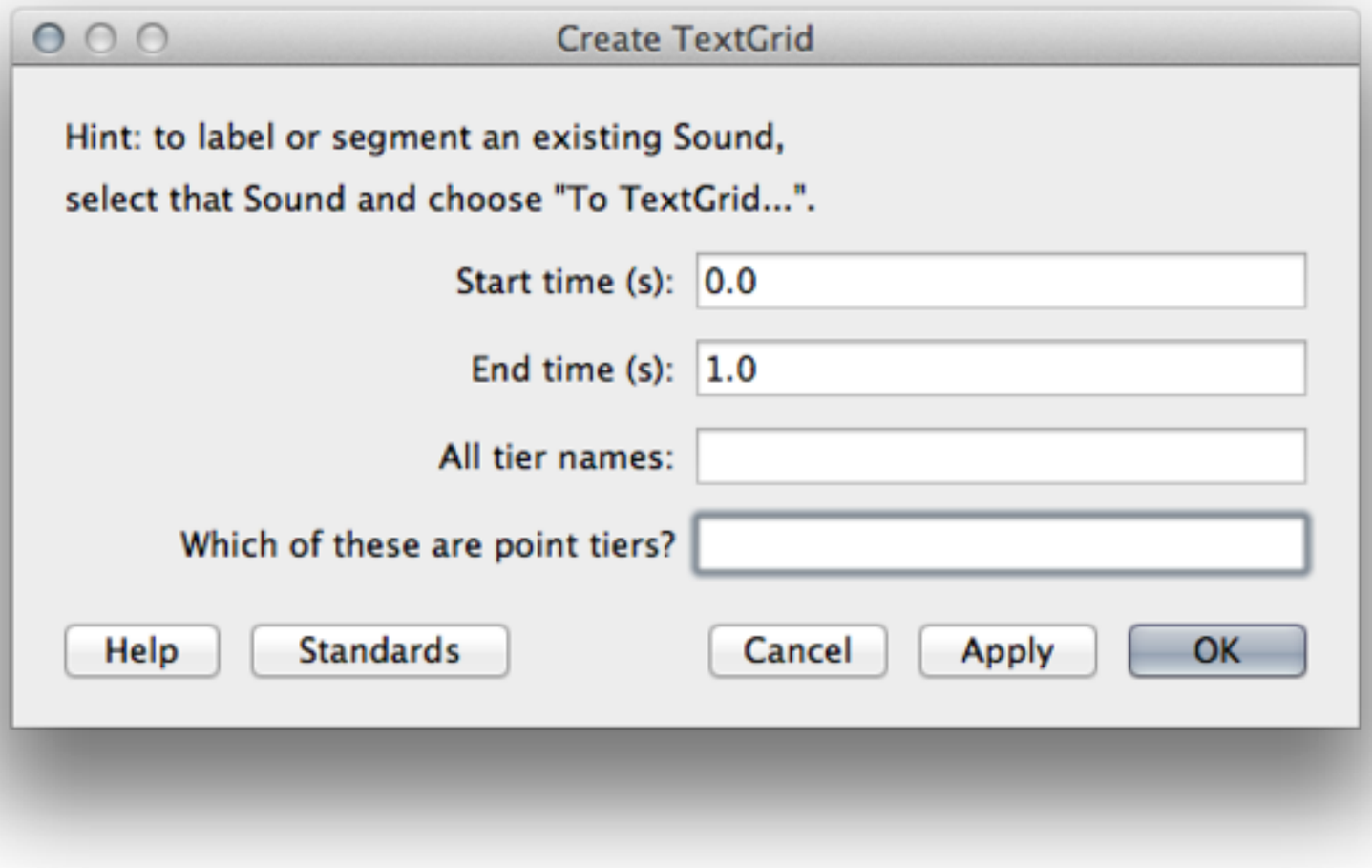
Section 2: Textgrids & Loops

TextGrid refresher

- A **Textgrid** is basically another kind of variable in Praat
- Instead of a simple string$ or number, though, a TextGrid is a multidimensional array
- It allows us to store and access structured data (typically about an associated sound file)



Create a TextGrid manually



- Normally a TextGrid is associated with a Sound object
- But it doesn't have to be
- New --> Create TextGrid...

So what?

- TextGrids store your segmentation judgments
- They make it easy to visualize transcriptions alongside their associated Sound objects
- TextGrids also make it easy to query specific intervals or points in your recordings
- And with scripting you can automate this!

Exercise

1. Open a new ScriptEditor window and clear your history

2. Read your sound file from yesterday into a Sound object (or record it again if you didn't save it)

3. Create a new TextGrid for this sound object (Annotate --> To TextGrid...)

4. Name the tiers: `words` and `onsets`

5. Create `onsets` as a point tier and click OK

6. Select both the Sound object and the TextGrid object
    1. Macintosh: command + click
    2. Windows or Linux: control + click

7. Paste command history into your ScriptEditor window

Create a TextGrid

---

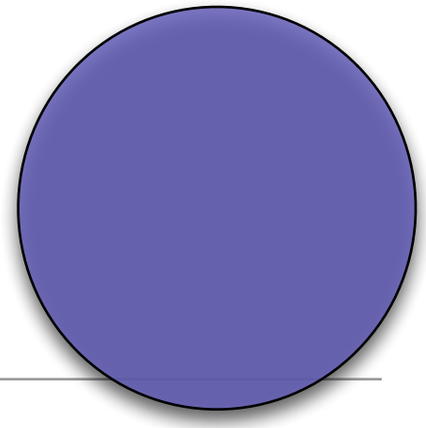
selectObject: "Sound untitled"
snd = selected: "Sound"
tg = To TextGrid: "words onsets", "onsets"

writeInfoLine: snd, tg
to View and Edit a TextGrid, you select two objects.
selectObject: snd
plusObject: tg

question: what hidden action must selectObject: be performing?

selectObject() ; try this command (parens needed)

Quick Exercise: Create some intervals



- In the TextGridEditor...

- Select (with the mouse) the Sound part of the display where the interval should be

- From the `Interval` menu, select `Add interval on tier 1 (or just press [return], or just type ⌘1)`

- Start typing to add a text label (the interval you just created is already implicitly selected)

- Save as a text file

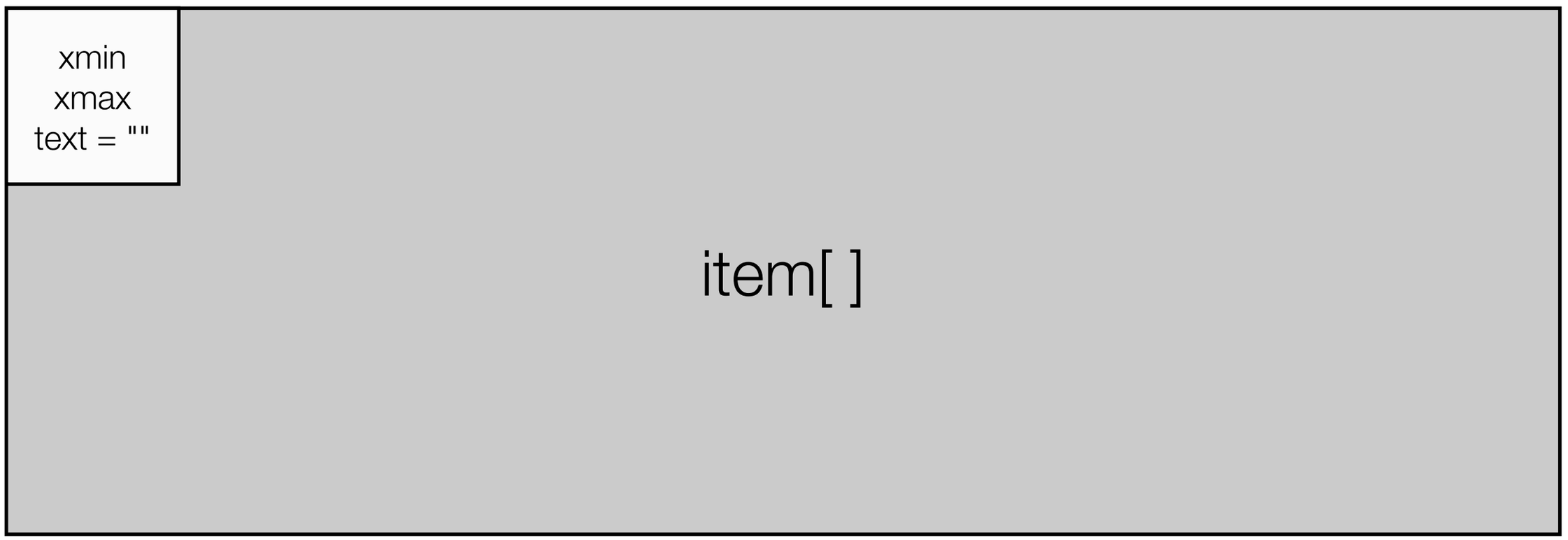
Envisioning TextGrid Structure

xmin xmax

item[ ]

Envisioning TextGrid Structure

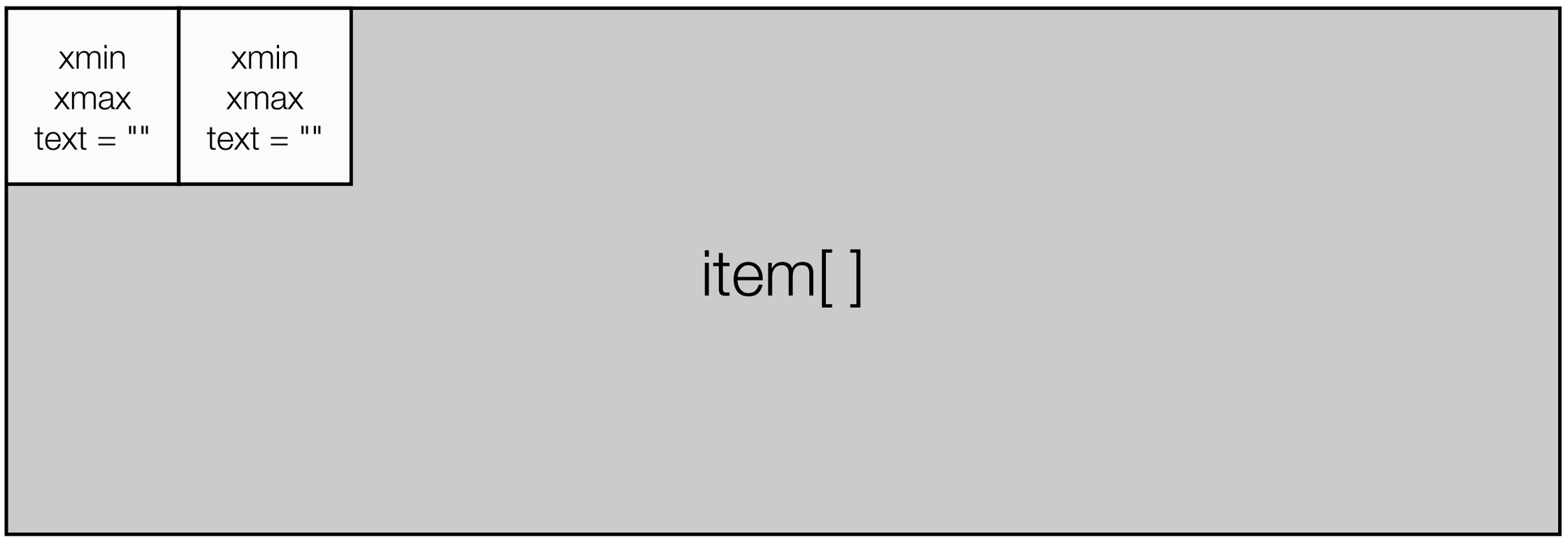
xmin xmax



Envisioning TextGrid Structure

xmin

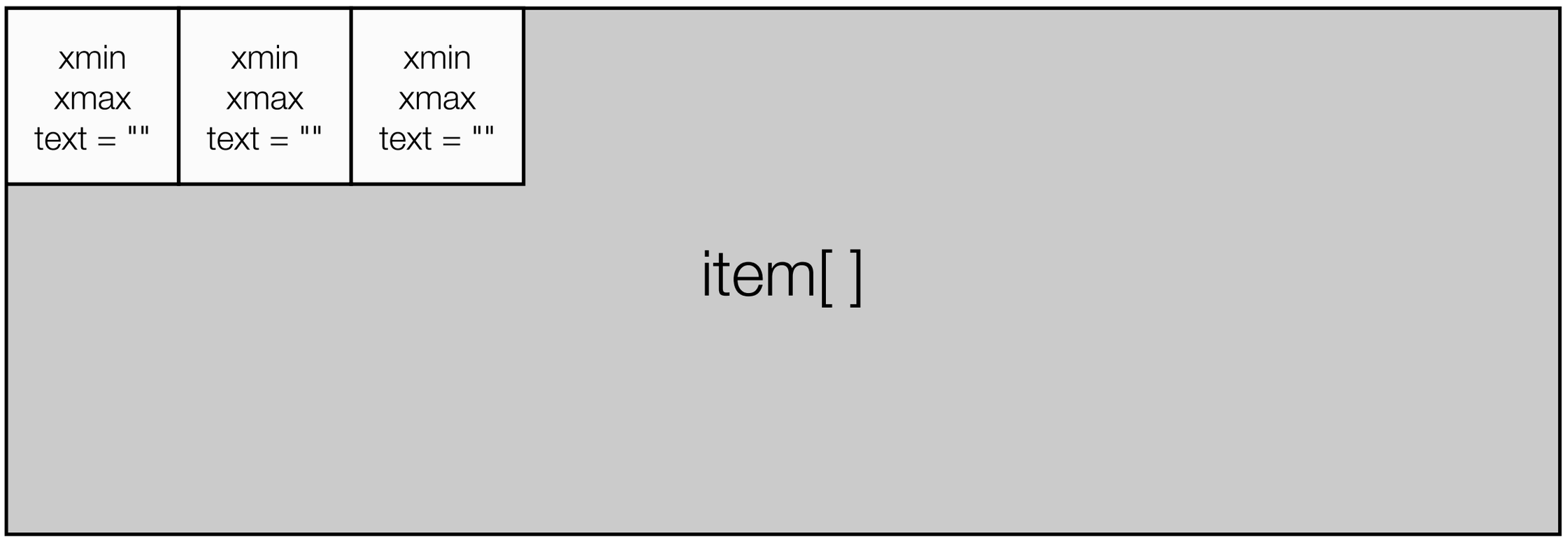
xmax



Envisioning TextGrid Structure

xmin

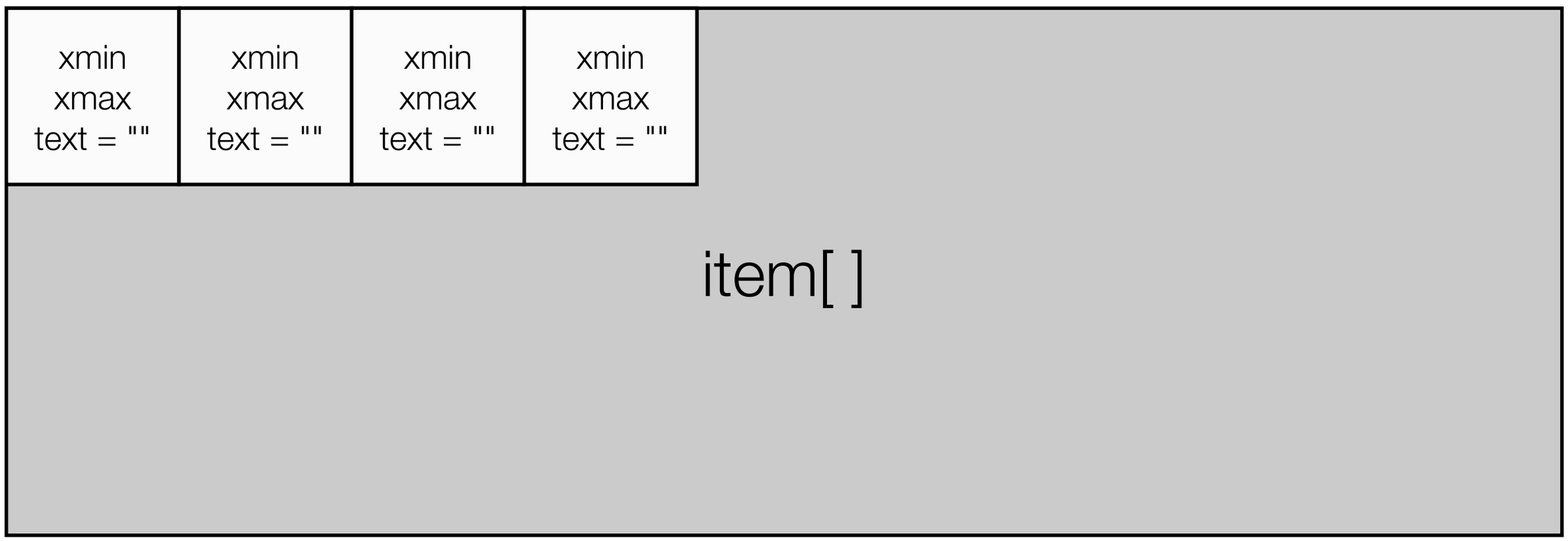
xmax



Envisioning TextGrid Structure

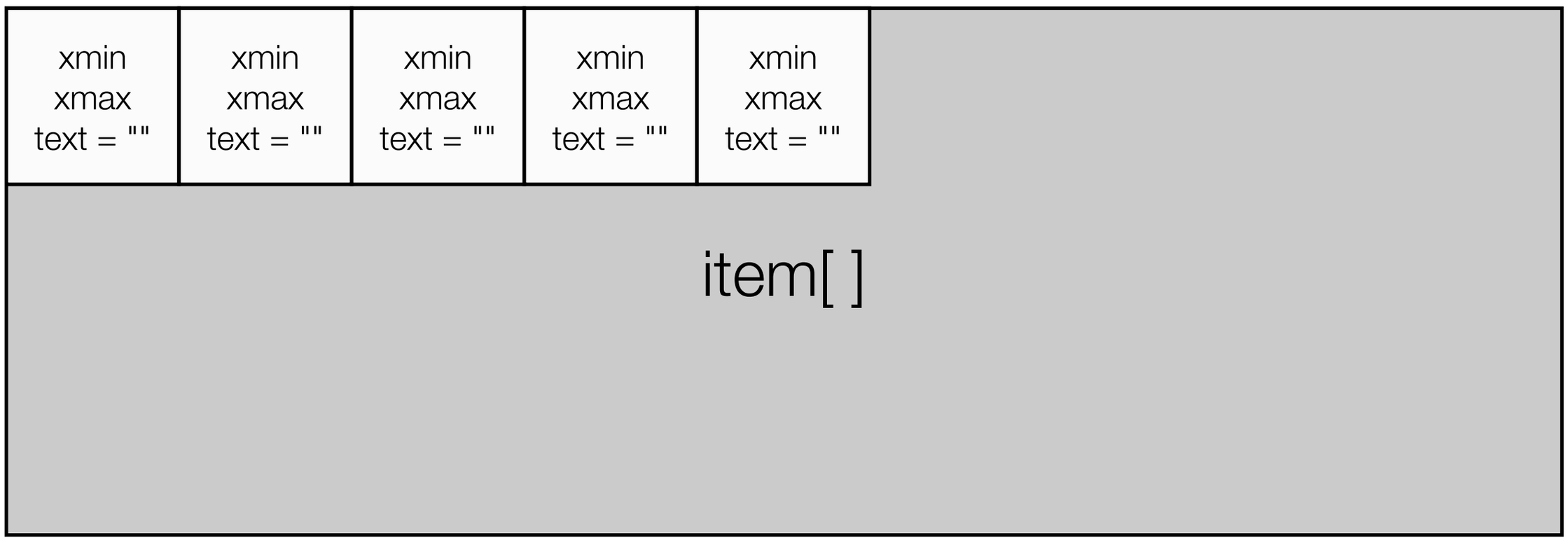
xmin

xmax



Envisioning TextGrid Structure

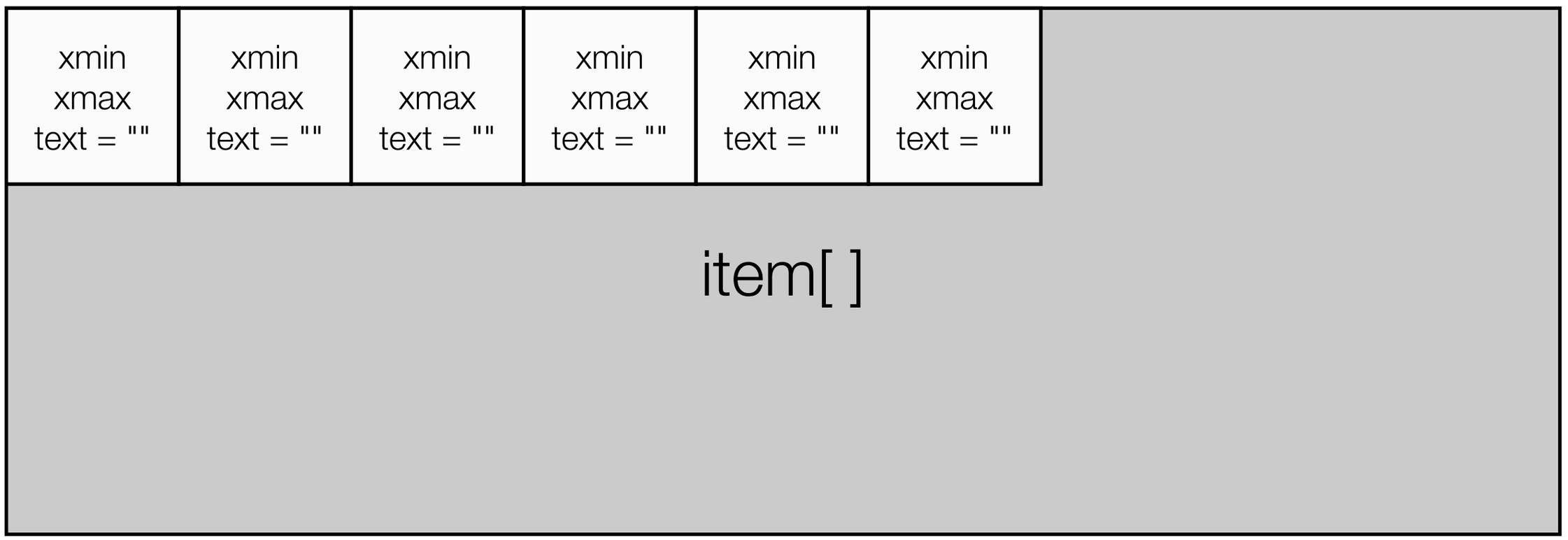
xmin xmax



Envisioning TextGrid Structure

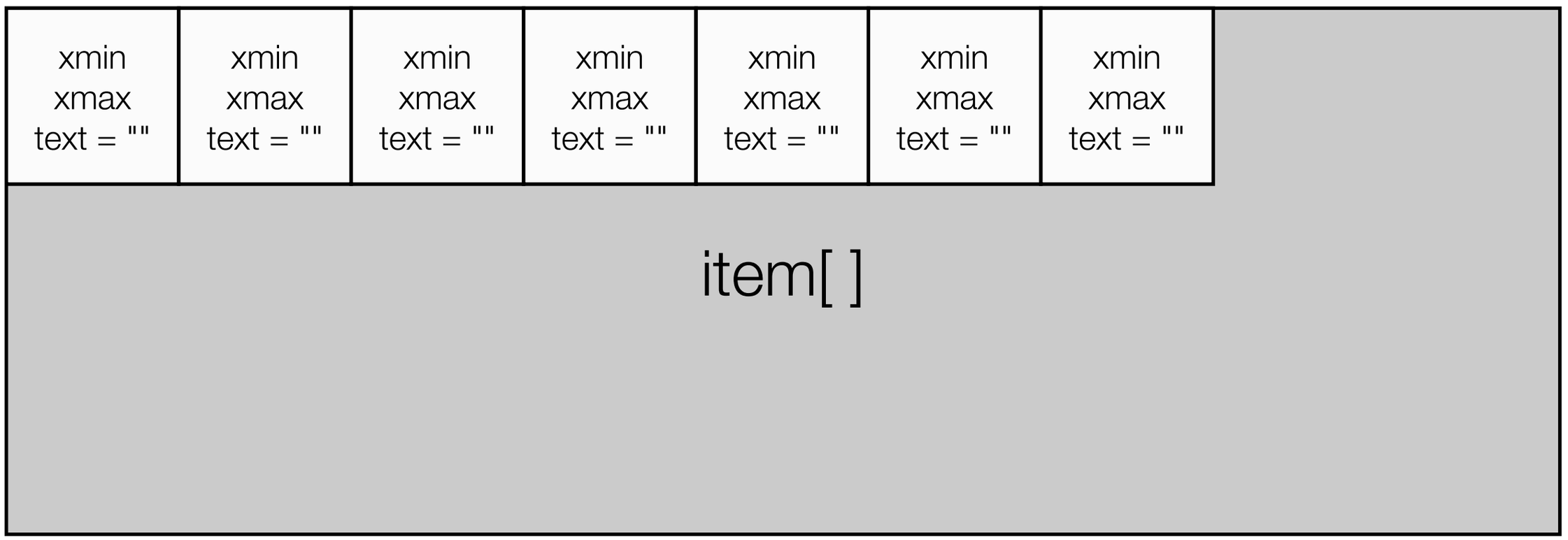
xmin

xmax



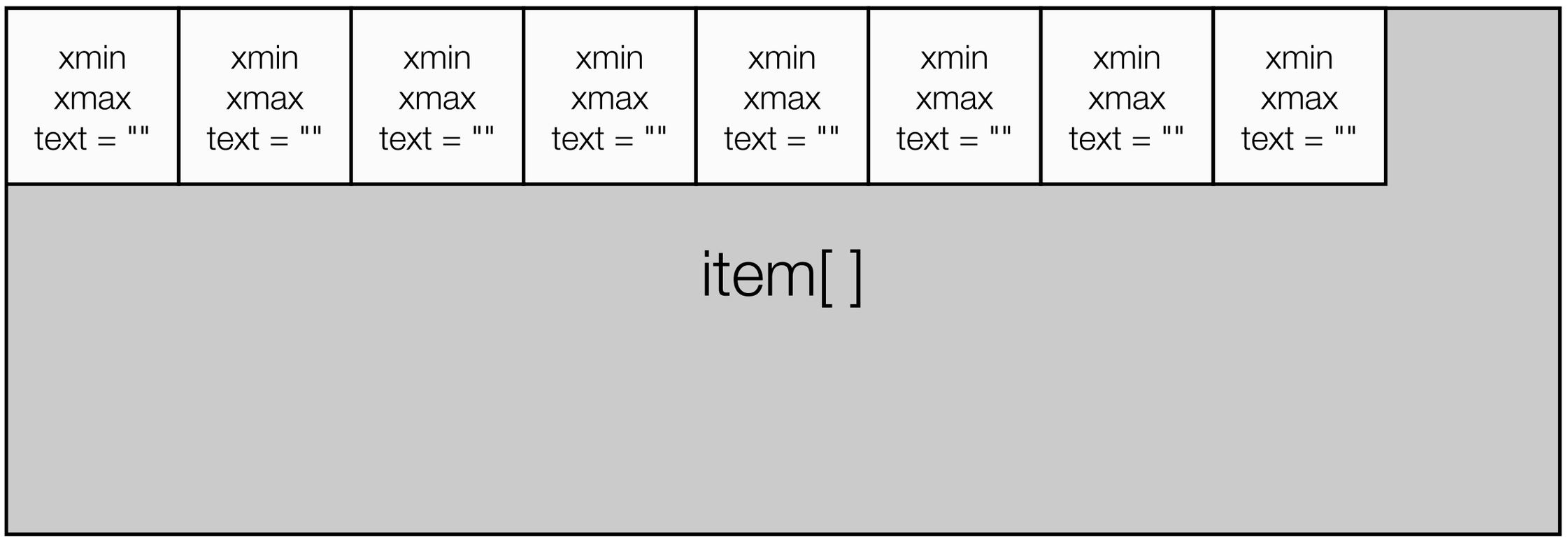
Envisioning TextGrid Structure

xmin xmax



Envisioning TextGrid Structure

xmin xmax



Envisioning TextGrid Structure

xmin xmax

xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""
item[ ]								

Envisioning TextGrid Structure

xmin xmax

xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""
xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""
xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""	xmin xmax text = ""

```
File type = "ooTextFile"
Object class = "TextGrid"

xmin = 0
xmax = 0.4
tiers? <exists>
size = 2
item []:
    item [1]:
        class = "IntervalTier"
        name = "words"
        xmin = 0
        xmax = 0.4
        intervals: size = 1
        intervals [1]:
            xmin = 0
            xmax = 0.4
            text = ""
item [2]:
    class = "TextTier"
    name = "onsets"
    xmin = 0
    xmax = 0.4
    points: size = 0
```

TextGrid contents

- A TextGrid is just a text file!
- **xmin** is the start of the time domain and **xmax** is the end
- **size** is the number of tiers
- Intervals in an IntervalTier also have **xmin**, **xmax** but also **text**

Extended Exercise 2: data from a TextGrid

1. From code.zip, open the “tobi” folder

2.Write a script to:

    1.Load the tobi.wav and tobi.TextGrid files into the Objects window

    2. Print the duration and number of tiers to the Info window

    3. Then for each tier, do the following:

        1. Print the tier number

        2. If this tier is an interval tier:

            1. Print the number of intervals

        3. Else (it is a point tier)

            1. Print the number of points

Ea 2 Solution (part 1 of 4)

```
# ea2-key.praat : read TextGrid file and report
# some attributes

tGrid = Read from file: "tobi/tobi.TextGrid"
totalDuration = Get total duration
writeInfoLine: "Total duration: ", fixed$(totalDuration, 2), "
seconds"

numberOfTiers = Get number of tiers
appendInfoLine: "Number of tiers is: ", numberOfTiers

# print out tier information one tier at a time
```

Ea 2 Solution (part 2 of 4)

```
name$ = Get tier name: 1
appendInfoLine: tab$, "Tier 1: """, name$, """"

if do( "Is interval tier...", 1 )
     intervals = Get number of intervals: 1
     appendInfoLine: tab$, tab$, "there are ", intervals,
     ..." intervals."
else
     points = Get number of points: 1
     appendInfoLine: tab$, tab$, "there are ", points,
     ..." points."
endif
```

Ea 2 Solution (part 3 of 4)

```
name$ = Get tier name: 2
appendInfoLine: tab$, "Tier 2: """, name$, """"

if do( "Is interval tier...", 2 )
     intervals = Get number of intervals: 2
     appendInfoLine: tab$, tab$, "there are ", intervals,
     ..." intervals."
else
     points = Get number of points: 2
     appendInfoLine: tab$, tab$, "there are ", points,
     ..." points."
endif
```

Ea 2 Solution (part 4 of 4)

```
name$ = Get tier name: 3
appendInfoLine: tab$, "Tier 3: """, name$, """"

if do( "Is interval tier...", 3 )
     intervals = Get number of intervals: 3
     appendInfoLine: tab$, tab$, "there are ", intervals,
     ..." intervals." )
else
     points = Get number of points: 3
     appendInfoLine: tab$, tab$, "there are ", points,
     ..." points."
endif

removeObject: tGrid
```

[do this with a loop]

Praat Scripting

Declarations

Unknown Variable

- So far we've just assigned variables whenever we need them, but

- Praat will give you an "Unknown variable" error if you try to use a variable before you declare it.

- Try something like one of the following:

```
y = m * x + b

text$ = "Please say what this word is " + bit$
```

- What can we say about the errors?

Declaring Variables

- The way around this problem is to always **declare** each variable prior to using it. We could have fixed our line formula with something like:

```
deltaX = 2

deltaY = 4

b = 1

m = deltaY / deltaX

x = 8

y = m * x + b

writeInfoLine: "y equals ", y, " when x is ", x
```

Unknown Variable Example

- You'll also get this Unknown Variable error if you try to use a variable in a test prior to assigning it any value, e.g.

```
if length( variable$ )
    writeInfoLine: "It exists and it is """, variable$, """"
else
    writeInfoLine: "Variable has not been set yet"
endif
```

Quick Exercise

- Please fix error.praat (you can download it with code.zip)
- Can you cause it to print both of its possible messages?

```
# error.praat - generates an error
if length( variable$ )
    writeInfoLine: "It exists and it is """, variable$, """"
else
    writeInfoLine: "Variable has not been set yet"
endif
```

Praat Scripting

Introduction to loops

Lather, rinse, repeat

- This seemingly simple program, found on most shampoo bottles, is actually stunningly complex.

- People are smart enough to handle the hidden complexity, but computers are insanely stupid.

- The hard part of looping is always going to be figuring out how to make your task repeatable.

Iteration

- Iteration is simply the ability for a programming language to specify that a block of code be executed over and over

- We will see two types of loops:

- **list iteration:** In Praat, **for** loops allow us to say "repeat this block for each number from n to k"

- **conditional iteration:** In Praat, **while** and **until** loops allow us to specify a test condition and either repeat our block of code until that test condition is false or until it is true.

List Iteration

- e.g. for each house on the street from 980 to 1422
    - knock on door,
    - yell, "trick or treat",
    - receive candy,
    - say, "thank you!"
- Notice that there are three parts:
    1. Variable representing the thing you are operating on (here the 'house'),
    2. A starting number, and
    3. An ending number

List Iteration

- The variable will be a numeric variable.
- The starting number can be any Praat expression that returns a number
- The ending number can also be any Praat expression that returns a number

100 coin tosses with ‘for’

```
# coins.praat

heads = 0
tails = 0

for toss from 1 to 100
     if randomInteger(1, 2) = 1
          heads = heads + 1
     else
          tails = tails + 1
     endif
endfor

writeInfoLine: "Praat simulated ", heads, " heads & ",
...tails, " tails."
```

Loading Files

```
# loader.praat - load all of the (lowercase) .wav files
# from dir$ into the Objects list

dir$ = "stimuli/"
strings = Create Strings as file list: “list", dir$ + "/*.wav"

numberOfFiles = Get number of strings

for ifile to numberOfFiles
    selectObject: strings
    fileName$ = Get string: ifile
    Read from file: dir$ + "/" + fileName$
endfor

removeObject: strings
```

Exercise: Plotting Slope & Intercept (1/2)

```
# slope.praat - use slope + intercept to draw points and line

# set up Picture window and draw x & y axes
Erase all
Select outer viewport: 0, 6, 0, 6
Axes: 0, 30, 0, 30
Draw line: 0, -30, 0, 30
Draw line: -30, 0, 30, 0

# clear the info window
writeInfoLine()

# continues on next slide...
```

Plotting Slope & Intercept (2/2)

```
# slope and intercept formula for a line
deltaX = 2
deltaY = 4
b = 1
x = 0
m = deltaY / deltaX
y = m * x + b

appendInfoLine: "y equals ", y, " when x is ", x

# challenge 1: can you make this plot 10 circles?
Draw circle: x, y, 0.5

# challenge 2: can you draw a line through the points?
# Draw line: startingX, startingY, finalX, finalY

# challenge 3: what if you wanted the points and line to
# appear to continue to infinity?
```

Extended Assignment 3:

- Part 1: modify the tobi reporting script you wrote earlier to use a loop
- Part 2: (optional, outside of class) write a script to:

    1. Read through a directory of files (in code/ea3Files)
    2. Load each wav file into the Objects window
    3. Check if each wav file has a corresponding TextGrid file
        1. If it does:
            1. open the TextGrid file as a TextGrid object
        2. if it does not:
            1. create the TextGrid object (see the existing TextGrid for tier names and types)
            2. Save it to a text file
    4. E-mail me the script for comments (we will do this together in class tomorrow)

Praat Scripting

Conditional Looping

Conditional Iteration

- e.g. while the sign is a red hand, continue not to walk

- The test for a loop can be any of the tests that we have used for conditionals so far

- In other words... any Praat expression or variable that will be 0 when false and something else when true

Conditional Iteration

- Praat lets you put your test at the end or the beginning of your loop...
    - **repeat/until *test*** always executes at least once, ends when the test is true
    - **while *test*/endwhile** may execute zero times!
- Which one you use when is often a question of taste (and readability)

Repeat Until Boxcars

```
# simulate a pair of six-sided dice until we roll two sixes
throws = 0

repeat
      pips = randomInteger(1, 6) + randomInteger(1, 6)
      throws = throws + 1
until pips = 12  ; total is 12 when both dice are sixes

writeInfoLine: "It took ", throws, " trials to reach ", pips,
..." with two dice."
```

Exercise: while not boxcars

```
# can you modify this code (boxcars.praat) to use
# while/endwhile instead of repeat/until?

throws = 0

repeat
     pips = randomInteger(1, 6) + randomInteger(1, 6)
     throws = throws + 1
until pips = 12  ; total is 12 when both dice are sixes

writeInfoLine: "It took ", throws, " trials to reach ", pips,
" with two dice."
```

While not Boxcars (solution)

```
throws = 0
pips = 0

while pips <> 12 ; total is 12 when both dice are sixes
      pips = randomInteger(1, 6) + randomInteger(1, 6)
      throws = throws + 1
endwhile
```

```
writeInfoLine: "It took ", throws, " trials to reach ", pips,
..." with two dice."
```

Return to TextGrids

Exercise: enloopification

- Please convert your solution to ea2 (the tobi reporting script) to use a single loop (feel free to iterate a list or to use a conditional loop) instead of those three repeated blocks. [reminder]

- You can find ea2-key.praat in code.zip if you would rather start with my version.

Enloopification (a solution)

```
for tier from 1 to numberOfTiers
     name$ = Get tier name: tier
     appendInfoLine: tab$, "Tier ", tier, ": """,
     ...name$, """"

     if do ( "Is interval tier...", tier )
          intervals = Get number of intervals: tier
          appendInfoLine: tab$, tab$, "there are ", intervals,
          ..." intervals."
     else
          points = Get number of points: tier
          appendInfoLine: tab$, tab$, "there are ", points,
          ..." points."
     endif
endfor

appendFile: "foo.txt", info$()
```

Enloopification (a solution)

```
for tier from 1 to numberOfTiers
     name$ = Get tier name: tier
     appendInfoLine: tab$, "Tier ", tier, ": """,
     ...name$, """"

     if do ( "Is interval tier...", tier )
          intervals = Get number of intervals: tier
          appendInfoLine: tab$, tab$, "there are ", intervals,
          ..." intervals."
     else
          points = Get number of points: tier
          appendInfoLine: tab$, tab$, "there are ", points,
          ..." points."
     endif
endfor

appendFile: "foo.txt", info$()
```

Save info window contents to text file!

Praat Scripting

Nested Loops

Nested loops

- Imagine you are decorating cookies. If you have multiple trays of cookies, your solution will be a nested loop!
- For each cookie tray:
  - move tray to table
  - while there are undecorated cookies on the tray
    - take a cookie
    - spread white frosting
    - let cool slightly
    - add ears, eye, and nose
    - place on drying rack
- move empty tray to sink

Exercise: Data from a TextGrid

- Please open the three files in code/tobi/
- Please modify the loopified version of ea2 to add the following:
  - for each interval tier
    - if the tier is called "words"
      - for each interval on the tier
        - if the interval has a label set
          - write that label to the info window
          - get the start time and end for the labeled interval
          - write the start and stop times to the Info window

Exercise: Data from a TextGrid (optional)

- Optionally, modify the loopified version of ea2 to add the following:
  - for each point tier
    - for each point
      - get the time and label of the point
      - write point, time, and label to the Info window

Extended Exercise: tokenize the Sound object

- Once you have the first exercise completed
- Try to use the label along with start and stop times from the "words" tier to save the sound associated with that interval to separate wav files.
- Hint: select a Sound object and see what the command: Convert --> Extract part... can do.
- Hint 2: Save as WAV file: "tobi/" + label$ + ".wav"

Praat Scripting

Goodies

Form (manual)

- Forms are a powerful way for your script to ask for user input at the beginning of runtime

- You will use the **`form`**/**`endform`** keywords to introduce a form block

- This form block can be anywhere in your source file (it will always run first!)

- Form fields have three parts: a field name, a field type, and a default value

- These form fields look like variables (and eventually become variables) **but they are not variables**.

- NOTE: code in **form**/**endform** blocks behaves differently. Read the manual.

Field Names

- The first evidence that these are not Praat variables is that they can begin with an upper case letter

- These upper case letters will be replaced with lower case letters in your script e.g.:

```
form Greet someone
    text Greeting Hello, world!
endform

writeInfoLine: greeting$
```

Field Names

- The first evidence that these are not Praat variables is that they can begin with an upper case letter

- These upper case letters will be replaced with lower case letters in your script e.g.:

```
form Greet someone
    text Greeting Hello, world!
endform

writeInfoLine: greeting$
```

Field Names

- The first evidence that these are not Praat variables is that they can begin with an upper case letter

- These upper case letters will be replaced with lower case letters in your script e.g.:

```
form Greet someone
    text Greeting Hello, world!
endform

writeInfoLine: greeting$
```

Exercise

- Please open yahw.praat from code.zip

1. Try adding double quotes to the title. Given your previous experiences with Praat, what *should* happen? What actually happens?
2. Try adding double quotes to `Hello, world!`. Again, what should happen and what does happen?

Field Types (manual)

- As you know, Praat has numeric and string variables.
    - numeric variables can contain integer or floating point
    - string variables can contain a single character or a copy of "Remembrance of Things Past"
- But sometimes we might want to further restrict what kind of data the form will accept so that Praat can ask them to fix it.
- For example, if only a whole number value will do, we can prompt for an `integer` which will require the user to enter a whole number but will be available to our script as a numeric variable

Field Types (manual)

- numeric
  - **`real`** `variable initialValue` real numbers
  - **`positive`** `variable initialValue` positive real numbers
  - **`integer`** `variable initialValue` whole numbers
  - **`natural`** `variable initialValue` positive whole numbers

- string
  - **`word`** `variable initialValue` a string without spaces
  - **`sentence`** `variable initialValue` any short string
  - **`text`** `variable initialValue` any possibly long string

- selection
  - **boolean** variable initialValue a check box
  - **choice** variable initialValue radio buttons

- **`comment`** `text` a line with any text. Does not become a variable!

Field types

- Let's try this together:

1. In yahw.praat, try changing `text` to each of the other text field types available in Praat forms.
2. How does this change the form that pops up?
3. How does this change the Info Window output from your script?
4. Now add a field called `Natural` asking for a natural number. What happens if you give it a zero?
5. What happens if you assign 0 to `natural` later in the script?

Form (example)

```
# formTone.praat - prompt the user for freq & gain
# from the Praat help manual (modified)

form Play a sine wave
    positive Sine_frequency_(Hz) 377
    positive Gain_(0..1) 0.3 (= not too loud)
endform
```

```
toneID = Create Sound as pure tone:
... "sine" + string$( sine_frequency ), 1, 0, 1, 44100,
... sine_frequency, gain, 0.01, 0.01)

Play
removeObject: toneID
```

Exercise: Math Practice

- Please open formTone.praat from code.zip

1. Add a form field to allow the user to specify the tone's duration.
2. 377 Hz is, by convention, an F.
    - The frequency difference between a note and the note one whole step above it is equal to the frequency of the starting `note x` $2^{1/6}$
    - The difference between a note and the note one half step above it is `note x` $2^{1/12}$
    - A major scale follows the pattern of intervals: `whole, whole, half, whole, whole, whole, half`
    - Can you modify formTone.praat to use a **for** loop and conditional to play a full major scale?
    - Can you make it also play a descending scale (without regenerating the tones)?

Exercise: divide and conquer

- The first step in software engineering is to make sure you understand the problem clearly enough to break it down into steps

- This allows you to solve the parts of the problem you understand (that are easy) while working toward the parts that are more challenging!

- How many notes does your script need to play? What does that mean for the kind of solution you devise?

- You have a working script that can play a note. First turn it into a working script that can play 8 notes.

- When you have 8 notes, figure out what has to change (from note to note) to get to 8 whole step intervals.

- From there, what has to change to add the half step intervals?

Scale sample solution (1)

```
# scale.praat - prompt the user for freq & gain
# then play a major scale: WWHWWWH

form Play a major scale
    positive Sine_frequency_(Hz) 377
    positive Gain_(0..1) 0.5 (= not too loud)
    positive Duration_(seconds) 0.5
endform

for step from 1 to 8
    if step = 1
        freq = sine_frequency
    elsif step = 4 or step = 8
        freq = freq * 2^(1/12)
    else
        freq = freq * 2^(1/6)
    endif
```

Scale sample solution (2)

```
      toneID = Create Sound as pure tone:
      ... "sine" + string$( freq ), 1, 0, duration, 44100,
      ... freq, gain, 0.01, 0.01

      Play
endfor
```

```
for tone from 1 to 8
      selectObject: toneID
      Play
      removeObject: toneID
      toneID = toneID - 1
endfor
```


DIFFICULT PRAAT EXERCISE
NAILED IT!
memegenerator.co

Optional Challenge: Keys

- Offer a set of radio buttons to let the user choose a specific starting note. e.g.
  - A = 440
  - A# = 466.2
  - B = 493.9
  - C = 523.3
  - C# = 554.4
  - D = 587.3
  - D# = 622.3
  - E = 659.3
  - F = 698.5
  - F# = 740.0
  - G = 784.0
  - G# = 830.6

Pause (manual)

- beginPause/endPause is a pair of keywords in Praat that allow you to prompt the user for input at any time in your script

- **beginPause/endPause** seems to do roughly the same thing as **form/endform**, but they're conceptually rather different

- Let's take a look at the manual and see... http://www.fon.hum.uva.nl/praat/manual/Scripting_6_6__Controlling_the_user.html

Remember this?

```
# randomGuess.praat : have the user guess a random number
num = randomInteger(1, 10)

form Guess a number
     comment I am thinking of a number between 1 and 10.
     integer Guess
endform

reply$ = "Your guess (" + string$( guess ) + ") is "

if guess = num
   reply$ = reply$ + "correct! Yay!"
elsif guess > num
   reply$ = reply$ + "too high."
else
   reply$ = reply$ + "too low."
endif

appendInfoLine: reply$
```

Exercise

- Please convert randomGuess.praat so that it:

    1. Pauses to prompt the user for a guess at runtime and

    2. Continues to prompt until the user guesses correctly

Praat Scripting

Using others' code

Using others' code

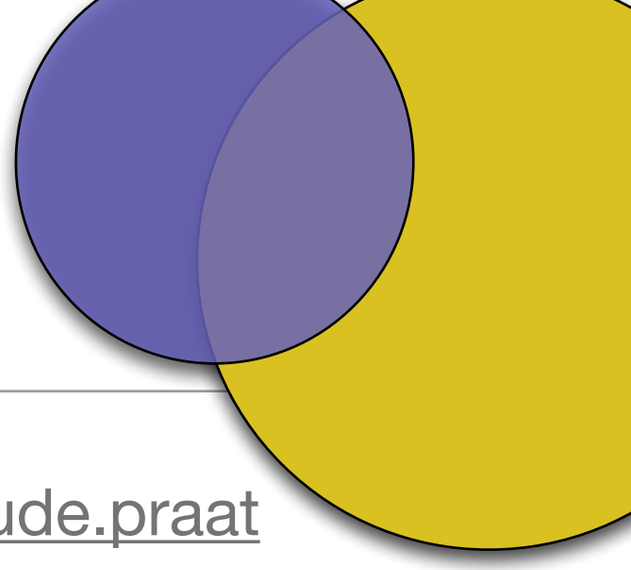
- One of the dirty secrets of software engineering is that most of a programmer's time is spent editing and maintaining existing code

- Writing a new program from scratch is comparatively rare

- While learning to program and using to learn Praat in this class, we have also been practicing the skills and habits of using others' code

1. Read every line and make sure you understand what it does (and whether it is necessary), “does this do what I need it to?”
2. Read critically! ask yourself, "does this do what the author thinks it does?" and "how could I do it better?"
3. Don't be afraid to reformat and debug
4. Double-check a sample of the output

Exercise: Understanding/Debugging



- http://www.phon.ucl.ac.uk/home/yi/praat/_normalize_amplitude.praat
    - This script normalizes all of the sound files in a directory, let's make sure we can understand it in terms of all three domains of knowledge we have been considering (linguistic, praat, and programming

Editor windows

- One of the behaviors available to some object types is to launch an editor window

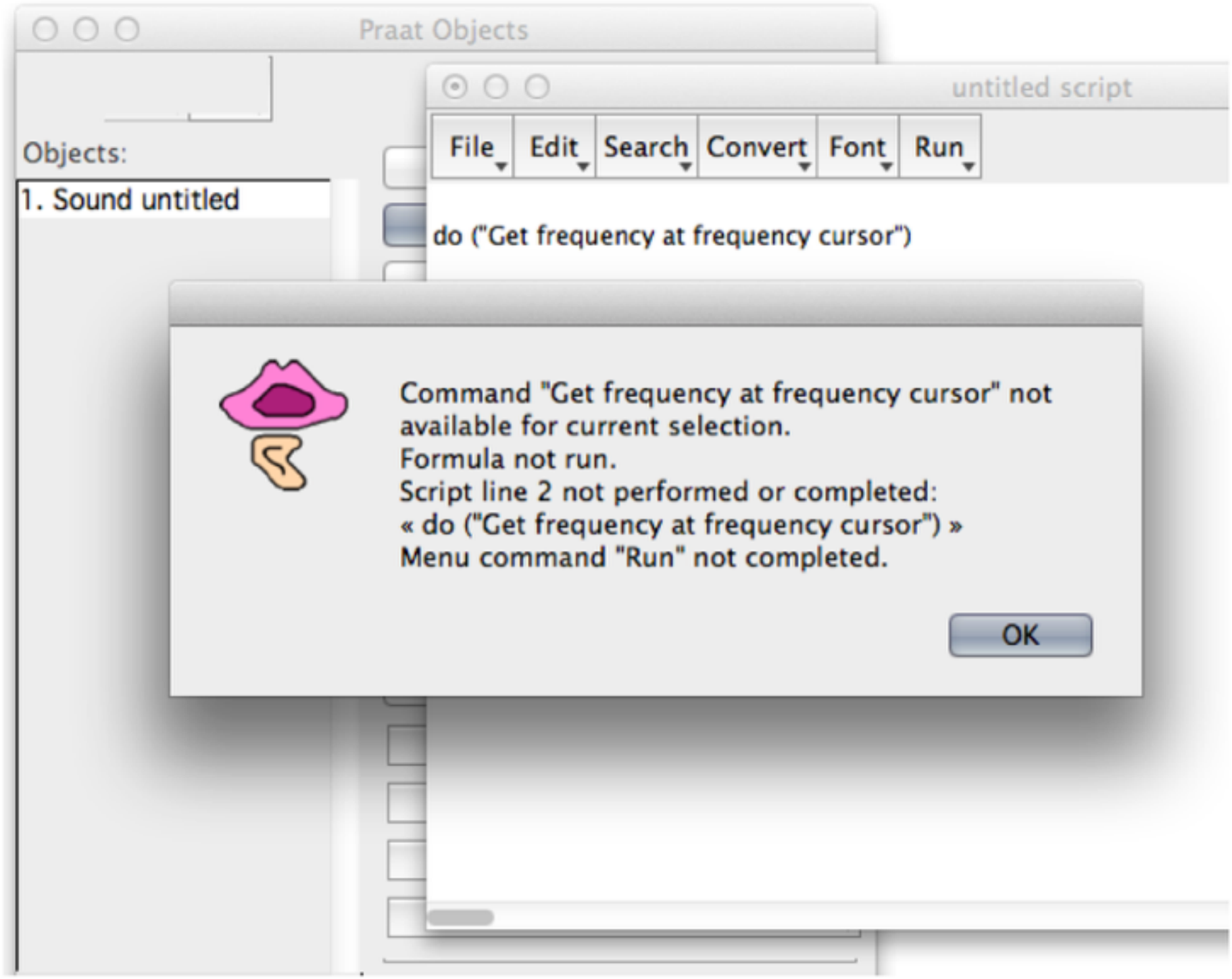
- The ones you will probably encounter most often are the SoundEditor and the TextGridEditor, but there are many others (help: Editors)

- You *can* control most editor window functions with a script (but normally you don't want to!)

- A very common source of runtime errors in Praat is attempting to issue an Editor window command from the Objects window (or vice versa).

- The error looks like this…

SoundEditor command from Objects Window



Scripting the editor

- Sometimes, you might want to write a script that will control one of Praat's 13 editors with a script.

- This makes the most sense when the script will become a new button or menu command.

- Let's look at draw_sprectrum_from_selection.praat by Mietta Lennes

- But first, something weird...

The ScriptEditor family

- Praat doesn't have just a single ScriptEditor, it has as many ScriptEditors as it has Editors + 1 (so 14).

- The main ScriptEditor (the one we've been using) can add commands to the Objects window or the Picture window.
    - Go to Praat --> New Praat Script --> File and see the selections "Add to fixed menu..." and "Add to dynamic menu..."

- The editor-specific ScriptEditors can add scripts to the menus in that editor.

Exercise: Extending Praat

- Open JFK_Moon_Rice.wav from code/sounds/
- Select the sound and open the SoundEditor window.
- In the SoundEditor, choose File --> Open editor script... to open Miette's draw_sprectrum_from_selection.praat
- In the ScriptEditor, use "Add to menu..." to add this command to the Spectrum menu. Call it something like "Draw spectrum from selection"
- Now close and reopen the SoundEditor window to use the menu command

draw_spectrum_from_selection.praat

```
# This script will draw a spectrum from a 40 ms window around
the cursor in the editor window.
#
# The original version of the script can be found in the
scripting tutorial of the built-in Praat manual.
# However, the original script does not work with the new
Praat versions; this one does. :-)
#
# 11.3.2002 Mietta Lennes
```

- Let's look at the rest of the code in Praat's SoundEditor's ScriptEditor window...

Exercise: TextGrid Editor

- Create a TextGrid for the JFK sound object
- Open the Sound and TextGrid objects together.
- Why isn't the FFT script in the Spectrum menu?
- Can you add it?

Crosswhite's duration logger

- Katherine Crosswhite has a large number of slightly old but really useful scripts available online.

- Duration_logger.praat is an example of a widely-available script that many have used (and still use)

- You have written essentially this same script!

- Let's review the code now (in code/duration_logger.praat)

duration_logger.praat

```
###Description of this script
##  This script the duration of all intervals marked in tier 1
with non-null lables.  Durations, in milliseconds
##   will be written to a log file called "duration-log.txt",
which you will be able to find in the same directory
##  holding your sound files after you run the script.
##  To run this script, you will have to have a bunch of sound
files with accompanying text grids.  Actually,
##   the sound files are superfluous, since they are not
needed to get duration. You could run this script
##   on a directory that had only text grids in it, but it is
more likely that you will have both sounds and text grids.
##   The sound files will be ignored. The locations of things
to be measured must be marked in tier 1 of the textgrid.
##   Anything with a non-null label in that tier will be
logged.
###End of description
```

Resources & Links

- An unordered, incomplete list of useful sources of more Praat online
- **Bartek Plichta** - scripts, instructions, and step-by-step how to videos
- Mark Antoniou - a handful of very useful scripts.
- **Pauline Welby** - some useful scripts and an introduction
- **Kyuchul Yoon** - many, many useful scripts and an introduction in Korean
- **Holger Mitterer** - very useful scripts, be sure to code review these first
- **Praat Script Archive** - a searchable index of several researchers' scripts

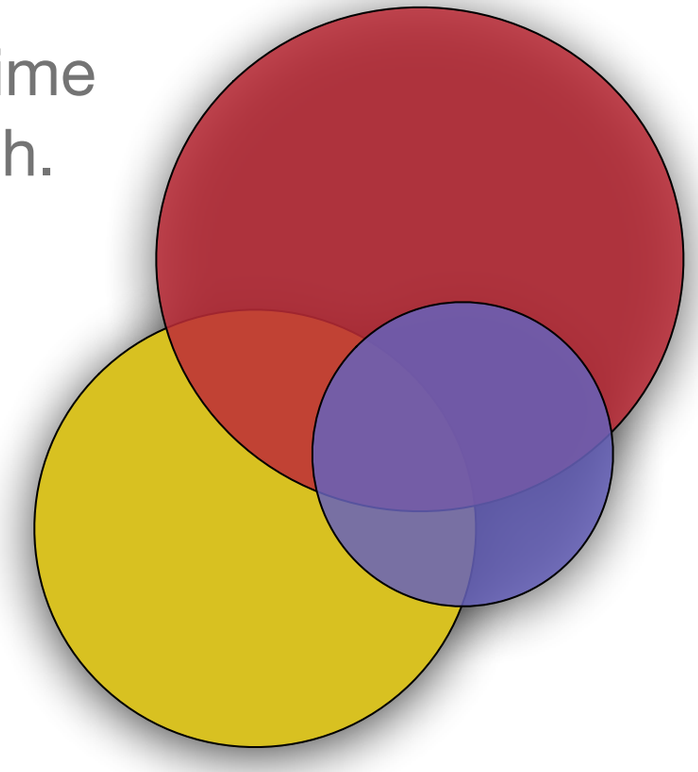
So what's the next step?

- Write a lot of Praat scripts. Minimally, write all of the scripts we've done in class yourself.

- Code review the scripts you already use (or new ones you find) and make sure you understand everything they do. Use the manual, ask friends, ask google, ask the praat-users list if you can't figure something out.

- Re-read the entire Praat manual. It will go faster this time and you'll learn things you missed the first time through.

- Teach someone else how to program Praat!



Praat Scripting

Nested Loops Solution

TextGrid Looping Solution

```
# nestedLoops.praat : read TextGrid file and report some attributes

tGrid = Read from file: "tobi/tobi.TextGrid"

totalDuration = Get total duration
writeInfoLine: "Total duration: ", fixed$( totalDuration, 2 ), " s"

nTiers = Get number of tiers
appendInfoLine: "Number of tiers is: ", nTiers

for tier to nTiers
    tierName$ = Get tier name: tier
    appendInfoLine: tab$, "Tier ", tier, ": """, tierName$, """"

    isTier = Is interval tier: tier

    if isTier = 1
        nIntervals = Get number of intervals: tier
        appendInfoLine: tab$, tab$, "there are ", nIntervals, " intervals."

        if tierName$ = "words"
            for interval from 1 to nIntervals
                intervalName$ = Get label of interval: tier, interval

                if intervalName$ <> ""
                                start = Get start point: tier, interval
                                end = Get end point: tier, interval

                                appendInfoLine: intervalName$, tab$, start, tab$, end
                endif
            endfor
        endif
    else ; this must be a point tier
        points = Get number of points: tier
        appendInfoLine: tab$, tab$, "there are ", points, " points."
        for point from 1 to points
            pointName$ = Get label of point: tier, point
            time = Get time of point: tier, point

            appendInfoLine: pointName$, tab$, time
        endfor
    endif
endfor
```

TextGrid Looping Solution zoomed (1/9)

```
# nestedLoops.praat : read TextGrid file and report
# some attributes

# see nestedLoops.praat in code.zip

tGrid = Read from file: "tobi/tobi.TextGrid"

totalDuration = Get total duration
writeInfoLine: "Total duration: ", fixed$( totalDuration, 2 ),
...” s"

nTiers = Get number of tiers
appendInfoLine: "Number of tiers is: ", nTiers

for tier to nTiers
        tierName$ = Get tier name: tier
        appendInfoLine: tab$, "Tier ", tier, ": """, tierName$, """"

        isTier = Is interval tier: tier

        if isTier = 1
            nIntervals = Get number of intervals: tier
```

TextGrid Looping Solution zoomed (2/9)

for tier to nTiers

```
tierName$ = Get tier name: tier
  appendInfoLine: tab$, "Tier ", tier, ": """, tierName$, """"

  isTier = Is interval tier: tier

  if isTier = 1
     nIntervals = Get number of intervals: tier
     appendInfoLine: tab$, tab$, "there are ", nIntervals, " intervals."

     if tierName$ = "words"
          for interval from 1 to nIntervals
               intervalName$ = Get label of interval: tier, interval

               if intervalName$ <> ""
                    start = Get start point: tier, interval
                    end = Get end point: tier, interval

                    appendInfoLine: intervalName$, tab$, start, tab$, end
               endif
          endfor
     endif
else ; this must be a point tier
     points = Get number of points: tier
     appendInfoLine: tab$, tab$, "there are ", points, " points."
     for point from 1 to points
          pointName$ = Get label of point: tier, point
          time = Get time of point: tier, point

          appendInfoLine: pointName$, tab$, time
     endfor
endif
```

endfor

TextGrid Looping Solution zoomed (2/9)

for tier to nTiers

```
tierName$ = Get tier name: tier
  appendInfoLine: tab$, "Tier ", tier, ": """, tierName$, """"

  isTier = Is interval tier: tier

  if isTier = 1
      nIntervals = Get number of intervals: tier
      appendInfoLine: tab$, tab$, "there are ", nIntervals, " intervals."

      if tierName$ = "words"
          for interval from 1 to nIntervals
              intervalName$ = Get label of interval: tier, interval

              if intervalName$ <> ""
                  start = Get start point: tier, interval
                  end = Get end point: tier, interval

                  appendInfoLine: intervalName$, tab$, start, tab$, end
              endif
          endfor
      endif
  else ; this must be a point tier
      points = Get number of points: tier
      appendInfoLine: tab$, tab$, "there are ", points, " points."
      for point from 1 to points
          pointName$ = Get label of point: tier, point
          time = Get time of point: tier, point

          appendInfoLine: pointName$, tab$, time
      endfor
  endif
```

endfor

TextGrid Looping Solution zoomed (3/9)

```
tierName$ = Get tier name: tier
appendInfoLine: tab$, "Tier ", tier, ": """, tierName$, """"

isTier = Is interval tier: tier

if isTier = 1
        nIntervals = Get number of intervals: tier
        appendInfoLine: tab$, tab$, "there are ", nIntervals, " intervals."
        if tierName$ = "words"
            for interval from 1 to nIntervals
                intervalName$ = Get label of interval: tier, interval

                if intervalName$ <> ""
                                start = Get start point: tier, interval
                                end = Get end point: tier, interval

                                appendInfoLine: intervalName$, tab$, start, tab$, end
                endif
            endfor
        endif
else ; this must be a point tier
        points = Get number of points: tier
        appendInfoLine: tab$, tab$, "there are ", points, " points."
        for point from 1 to points
            pointName$ = Get label of point: tier, point
            time = Get time of point: tier, point

            appendInfoLine: pointName$, tab$, time
        endfor
endif
```

TextGrid Looping Solution zoomed (3/9)

```
tierName$ = Get tier name: tier
appendInfoLine: tab$, "Tier ", tier, ": """, tierName$, """"

isTier = Is interval tier: tier

if isTier = 1
    nIntervals = Get number of intervals: tier
    appendInfoLine: tab$, tab$, "there are ", nIntervals, " intervals."

    if tierName$ = "words"
        for interval from 1 to nIntervals
            intervalName$ = Get label of interval: tier, interval

            if intervalName$ <> ""
                start = Get start point: tier, interval
                end = Get end point: tier, interval

                appendInfoLine: intervalName$, tab$, start, tab$, end
            endif
        endfor
    endif
else ; this must be a point tier
    points = Get number of points: tier
    appendInfoLine: tab$, tab$, "there are ", points, " points."
    for point from 1 to points
        pointName$ = Get label of point: tier, point
        time = Get time of point: tier, point

        appendInfoLine: pointName$, tab$, time
    endfor
endif
```

TextGrid Looping Solution zoomed (4/9)

```
nIntervals = Get number of intervals: tier
appendInfoLine: tab$, tab$, "there are ", nIntervals, " intervals."

if tierName$ = "words"
        for interval from 1 to nIntervals
            intervalName$ = Get label of interval: tier, interval

            if intervalName$ <> ""
                    start = Get start point: tier, interval
                    end = Get end point: tier, interval

                    appendInfoLine: intervalName$, tab$, start, tab$, end
            endif
        endfor
endif
```

TextGrid Looping Solution zoomed (4/9)

nIntervals = Get number of intervals: tier
appendInfoLine: tab$, tab$, "there are ", nIntervals, " intervals."

if tierName$ = "words"

```
for interval from 1 to nIntervals
    intervalName$ = Get label of interval: tier, interval

    if intervalName$ <> ""
                start = Get start point: tier, interval
                end = Get end point: tier, interval

                appendInfoLine: intervalName$, tab$, start, tab$, end
    endif
endfor
```

endif

TextGrid Looping Solution zoomed (5/9)

```
for interval from 1 to nIntervals
    intervalName$ = Get label of interval: tier, interval

    if intervalName$ <> ""
        start = Get start point: tier, interval
        end = Get end point: tier, interval

        appendInfoLine: intervalName$, tab$, start, tab$, end
    endif
endfor
```

TextGrid Looping Solution zoomed (6/9)

```
for interval from 1 to nIntervals
    intervalName$ = Get label of interval: tier, interval

    if intervalName$ <> ""
        start = Get start point: tier, interval
        end = Get end point: tier, interval

        appendInfoLine: intervalName$, tab$, start, tab$, end
    endif
endfor
```

TextGrid Looping Solution zoomed (6/9)

```
for interval from 1 to nIntervals
    intervalName$ = Get label of interval: tier, interval

    if intervalName$ <> ""
        start = Get start point: tier, interval
        end = Get end point: tier, interval

        appendInfoLine: intervalName$, tab$, start, tab$, end
    endif
endfor
```

Now we just need to go look at the ‘else’ block

TextGrid Looping Solution zoomed (7/9)

```
tierName$ = Get tier name: tier
appendInfoLine: tab$, "Tier ", tier, ": """, tierName$, """"

isTier = Is interval tier: tier

if isTier = 1
        nIntervals = Get number of intervals: tier
        appendInfoLine: tab$, tab$, "there are ", nIntervals, " intervals."
        if tierName$ = "words"
            for interval from 1 to nIntervals
                intervalName$ = Get label of interval: tier, interval

                if intervalName$ <> ""
                                start = Get start point: tier, interval
                                end = Get end point: tier, interval

                                appendInfoLine: intervalName$, tab$, start, tab$, end
                endif
            endfor
        endif
else ; this must be a point tier
        points = Get number of points: tier
        appendInfoLine: tab$, tab$, "there are ", points, " points."
        for point from 1 to points
            pointName$ = Get label of point: tier, point
            time = Get time of point: tier, point

            appendInfoLine: pointName$, tab$, time
        endfor
endif
```

TextGrid Looping Solution zoomed (7/9)

```
tierName$ = Get tier name: tier
appendInfoLine: tab$, "Tier ", tier, ": """, tierName$, """"

isTier = Is interval tier: tier

if isTier = 1
        nIntervals = Get number of intervals: tier
        appendInfoLine: tab$, tab$, "there are ", nIntervals, " intervals."
        if tierName$ = "words"
            for interval from 1 to nIntervals
                intervalName$ = Get label of interval: tier, interval

                if intervalName$ <> ""
                                start = Get start point: tier, interval
                                end = Get end point: tier, interval

                                appendInfoLine: intervalName$, tab$, start, tab$, end
                endif
            endfor
        endif
else ; this must be a point tier
        points = Get number of points: tier
        appendInfoLine: tab$, tab$, "there are ", points, " points."
        for point from 1 to points
            pointName$ = Get label of point: tier, point
            time = Get time of point: tier, point

            appendInfoLine: pointName$, tab$, time
        endfor
endif
```

TextGrid Looping Solution zoomed (8/9)

```
else ; this must be a point tier
        points = Get number of points: tier
        appendInfoLine: tab$, tab$, "there are ", points, " points."
        for point from 1 to points
            pointName$ = Get label of point: tier, point
            time = Get time of point: tier, point
            appendInfoLine: pointName$, tab$, time
        endfor
endif
```

TextGrid Looping Solution zoomed (9/9)

```
    else ; this must be a point tier
            points = Get number of points: tier
            appendInfoLine: tab$, tab$, "there are ", points, " points."
            for point from 1 to points
                pointName$ = Get label of point: tier, point
                time = Get time of point: tier, point
                appendInfoLine: pointName$, tab$, time
            endfor
    endif
```

Extended Exercise: tokenize the Sound object

- Once you have the first exercise completed
- Try to use the label along with start and stop times from the "words" tier to save the sound associated with that interval to separate wav files.
- Hint: select a Sound object and see what the command: Convert --> Extract part... can do.
- Hint 2: Save as WAV file: "tobi/" + label$ + ".wav"

Extracting WAVs solution

```
# extractWAVs.praat : read TextGrid file, report some attributes,
# save any labeled intervals on the "words" tier as wav files

wav = Read from file: "tobi/tobi.wav"
tGrid = Read from file: "tobi/tobi.TextGrid"

totalDuration = Get total duration
writeInfoLine: "Total duration: ", fixed$( totalDuration, 2 ), " s"

nTiers = Get number of tiers
appendInfoLine: "Number of tiers is: ", nTiers

for tier to nTiers
    tierName$ = Get tier name: tier
    appendInfoLine: tab$, "Tier ", tier, ": """, tierName$, """"

    if do ( "Is interval tier...", tier )
        nIntervals = Get number of intervals: tier
        appendInfoLine: tab$, tab$, "there are ", nIntervals, " intervals."
        if tierName$ = "words"
            for interval from 1 to nIntervals
                intervalName$ = Get label of interval: tier, interval

                if intervalName$ <> ""
                   start = Get start point: tier, interval
                    end = Get end point: tier, interval

                   appendInfoLine: intervalName$, tab$, start, tab$, end

                   selectObject: wav
                    part = Extract part: start, end, "rectangular", 1, "no"
                    Save as WAV file: "tobi/" + intervalName$ + "-" + string$( interval ) + ".wav"
                    removeObject: part
                    selectObject: tGrid
                endif
            endfor
        endif
    else ; this must be a point tier
        points = Get number of points: tier
        appendInfoLine: tab$, tab$, "there are ", points, " points."
        for point from 1 to points
            pointName$ = Get label of point: tier, point
            time = Get time of point: tier, point

            appendInfoLine: pointName$, tab$, time
        endfor
    endif
endfor

removeObject: wav
removeObject: tGrid
```

Extracting WAVs solution

```
# extractWAVs.praat : read TextGrid file, report some attributes,
# save any labeled intervals on the "words" tier as wav files

wav = Read from file: "tobi/tobi.wav"
tGrid = Read from file: "tobi/tobi.TextGrid"
```

```
totalDuration = Get total duration
writeInfoLine: "Total duration: ", fixed$( totalDuration, 2 ), " s"

nTiers = Get number of tiers
appendInfoLine: "Number of tiers is: ", nTiers

for tier to nTiers
    tierName$ = Get tier name: tier
    appendInfoLine: tab$, "Tier ", tier, ": """, tierName$, """

    if do ( "Is interval tier...", tier )
        nIntervals = Get number of intervals: tier
        appendInfoLine: tab$, tab$, "there are ", nIntervals, " intervals."
        if tierName$ = "words"
            for interval from 1 to nIntervals
                intervalName$ = Get label of interval: tier, interval

                if intervalName$ <> ""
                    start = Get start point: tier, interval
                    end = Get end point: tier, interval

                    appendInfoLine: intervalName$, tab$, start, tab$, end

                    selectObject: wav
                    part = Extract part: start, end, "rectangular", 1, "no"
                    Save as WAV file: "tobi/" + intervalName$ + "-" + string$( interval ) + ".wav"
                    removeObject: part
                    selectObject: tGrid
                endif
            endfor
        endif
    else ; this must be a point tier
        points = Get number of points: tier
        appendInfoLine: tab$, tab$, "there are ", points, " points."
        for point from 1 to points
            pointName$ = Get label of point: tier, point
            time = Get time of point: tier, point

            appendInfoLine: pointName$, tab$, time
        endfor
    endif
endfor

removeObject: wav
removeObject: tGrid
```

extractWAVs.praat : read TextGrid file, report some attributes,
save any labeled intervals on the "words" tier as wav files

wav = Read from file: “tobi/tobi.wav"
tGrid = Read from file: "tobi/tobi.TextGrid"

Extracting WAVs solution

```
# extractWAVs.praat : read TextGrid file, report some attributes,
# save any labeled intervals on the "words" tier as wav files

wav = Read from file: "tobi/tobi.wav"
tGrid = Read from file: "tobi/tobi.TextGrid"

totalDuration = Get total duration
writeInfoLine: "Total duration: ", fixed$( totalDuration, 2 ), " s"

nTiers = Get number of tiers
appendInfoLine: "Number of tiers is: ", nTiers

for tier to nTiers
    tierName$ = Get tier name: tier
    appendInfoLine: tab$, "Tier ", tier, ": """, tierName$, """"

    if do ( "Is interval tier...", tier )
        nIntervals = Get number of intervals: tier
        appendInfoLine: tab$, tab$, "there are ", nIntervals, " intervals."
        if tierName$ = "words"
            for interval from 1 to nIntervals
                intervalName$ = Get label of interval: tier, interval

                if intervalName$ <> ""
                   start = Get start point: tier, interval
                    end = Get end point: tier, interval

                   appendInfoLine: intervalName$, tab$, start, tab$, end

                   selectObject: wav
                    part = Extract part: start, end, "rectangular", 1, "no"
                    Save as WAV file: "tobi/" + intervalName$ + "-" + string$( interval ) + ".wav"
                    removeObject: part
                    selectObject: tGrid
                endif
            endfor
        endif
    else ; this must be a point tier
        points = Get number of points: tier
        appendInfoLine: tab$, tab$, "there are ", points, " points."
        for point from 1 to points
            pointName$ = Get label of point: tier, point
            time = Get time of point: tier, point

            appendInfoLine: pointName$, tab$, time
        endfor
    endif
endfor

removeObject: wav
removeObject: tGrid
```

Extracting WAVs solution

```
# extractWAVs.praat : read TextGrid file, report some attributes,
# save any labeled intervals on the "words" tier as wav files

wav = Read from file: "tobi/tobi.wav"
tGrid = Read from file: "tobi/tobi.TextGrid"

totalDuration = Get total duration
writeInfoLine: "Total duration: ", fixed$( totalDuration, 2 ), " s"

nTiers = Get number of tiers
appendInfoLine: "Number of tiers is: ", nTiers

for tier to nTiers
    tierName$ = Get tier name: tier
    appendInfoLine: tab$, "Tier ", tier, ": """, tierName$, """

    if do ( "Is interval tier...", tier )
        nIntervals = Get number of intervals: tier
        appendInfoLine: tab$, tab$, "there are ", nIntervals, " intervals."

        if tierName$ = "words"
            for interval from 1 to nIntervals
                intervalName$ = Get label of interval: tier, interval

                if intervalName$ <> ""
                    start = Get start point: tier, interval
                     end = Get end point: tier, interval

                    appendInfoLine: intervalName$, tab$, start, tab$, end

                    selectObject: wav
                     part = Extract part: start, end, "rectangular", 1, "no"
                     Save as WAV file: "tobi/" + intervalName$ + "-" + string$( interval ) + ".wav"
                     removeObject: part
                     selectObject: tGrid
                endif
             endfor
        endif
    else ; this must be a point tier
        points = Get number of points: tier
        appendInfoLine: tab$, tab$, "there are ", points, " points."
        for point from 1 to points
            pointName$ = Get label of point: tier, point
            time = Get time of point: tier, point

            appendInfoLine: pointName$, tab$, time
        endfor
    endif
endfor

removeObject: wav
removeObject: tGrid
```

```
selectObject: wav
part = Extract part: start, end, "rectangular", 1, "no"
Save as WAV file: "tobi/" + intervalName$ + "-" +
string$( interval ) + ".wav"
removeObject: part
selectObject: tGrid
```

Extracting WAVs solution

```
# extractWAVs.praat : read TextGrid file, report some attributes,
# save any labeled intervals on the "words" tier as wav files

wav = Read from file: "tobi/tobi.wav"
tGrid = Read from file: "tobi/tobi.TextGrid"

totalDuration = Get total duration
writeInfoLine: "Total duration: ", fixed$( totalDuration, 2 ), " s"

nTiers = Get number of tiers
appendInfoLine: "Number of tiers is: ", nTiers

for tier to nTiers
    tierName$ = Get tier name: tier
    appendInfoLine: tab$, "Tier ", tier, ": """, tierName$, """"

    if do ( "Is interval tier...", tier )
        nIntervals = Get number of intervals: tier
        appendInfoLine: tab$, tab$, "there are ", nIntervals, " intervals."
        if tierName$ = "words"
            for interval from 1 to nIntervals
                intervalName$ = Get label of interval: tier, interval

                if intervalName$ <> ""
                   start = Get start point: tier, interval
                    end = Get end point: tier, interval

                   appendInfoLine: intervalName$, tab$, start, tab$, end

                   selectObject: wav
                    part = Extract part: start, end, "rectangular", 1, "no"
                    Save as WAV file: "tobi/" + intervalName$ + "-" + string$( interval ) + ".wav"
                    removeObject: part
                    selectObject: tGrid
                endif
            endfor
        endif
    else ; this must be a point tier
        points = Get number of points: tier
        appendInfoLine: tab$, tab$, "there are ", points, " points."
        for point from 1 to points
            pointName$ = Get label of point: tier, point
            time = Get time of point: tier, point

            appendInfoLine: pointName$, tab$, time
        endfor
    endif
endfor

removeObject: wav
removeObject: tGrid
```

Extracting WAVs solution

```
# extractWAVs.praat : read TextGrid file, report some attributes,
# save any labeled intervals on the "words" tier as wav files

wav = Read from file: "tobi/tobi.wav"
tGrid = Read from file: "tobi/tobi.TextGrid"

totalDuration = Get total duration
writeInfoLine: "Total duration: ", fixed$( totalDuration, 2 ), " s"

nTiers = Get number of tiers
appendInfoLine: "Number of tiers is: ", nTiers

for tier to nTiers
    tierName$ = Get tier name: tier
    appendInfoLine: tab$, "Tier ", tier, ": """, tierName$, """

    if do ( "Is interval tier...", tier )
        nIntervals = Get number of intervals: tier
        appendInfoLine: tab$, tab$, "there are ", nIntervals, " intervals."
        if tierName$ = "words"
            for interval from 1 to nIntervals
                intervalName$ = Get label of interval: tier, interval

                if intervalName$ <> ""
                   start = Get start point: tier, interval
                    end = Get end point: tier, interval

                   appendInfoLine: intervalName$, tab$, start, tab$, end

                   selectObject: wav
                    part = Extract part: start, end, "rectangular", 1, "no"
                    Save as WAV file: "tobi/" + intervalName$ + "-" + string$( interval ) + ".wav"
                    removeObject: part
                    selectObject: tGrid
                endif
            endfor
        endif
    else ; this must be a point tier
        points = Get number of points: tier
        appendInfoLine: tab$, tab$, "there are ", points, " points."
        for point from 1 to points
            pointName$ = Get label of point: tier, point
            time = Get time of point: tier, point

            appendInfoLine: pointName$, tab$, time
        endfor
    endif
endfor

removeObject: wav
removeObject: tGrid
```

removeObject: wav

TextGrid Looping Solution (key points)

- For each tier & for each interval...

- Restore the correct selection if you change it (and remember, creating an object changes it)

- Be sure to clean up after yourself by removing objects when you're done with them.

    - And do this as soon as you can –don't wait until the end of the script.

- A simple thing like selecting a Sound object can have wide-reaching effects.