Praat scripting workshop

Session 1

Welcome!

Outline for today

- Reasons to automate
- Praat Basics
- Praat scripting - introductory concepts

Disclaimer

- This is a workshop on ~~how to script~~ learning how to teach yourself how to script

- Think cooking.

- So… you will learn *most* scripting on your own

  - We’re hoping to get you started.

Getting to know you

- What makes you interested in learning Praat scripting?
- Are you working on a project that would require Praat scripting?
- Have you used Praat scripts before? How did you use them?
- Do you use other software that requires coding? What's your experience with programming and scripting?
- Where do you find Praat scripts?

Data management

- Why it matters to think about how we manage our data?
  - Workflow!
    - Coordinated framework for all aspects of research
    - Planning, organizing and documenting research
    - Cleaning data
    - Analyzing data
    - Presenting results
    - Writing up results
    - Backing up data and materials

Inspired by Dr. J. Scott Long, "Reproducible Results and the Workflow of Data Analysis"

Why it matters to think about how we manage our data?

- Workflow!
  - Clumsy workflow, clumsy management, clumsy data.
  - Efficient workflow = productive use of your time (which means that you have more time to go out and relax and be human)
  - IT SAVES YOU TIME AND IT AND MAKES YOU A BETTER ANALYST

Why it matters to think about how we manage our data?

- Anticipate problems down the road.
  - Avoid having to put out fires all the time.
    - It's stressful.
    - It's time you should spent on getting to know your data.
- If you put the project on hold for a few weeks (or months or years!) it is easier to get back into it if you know what you have done and why (think time and mistakes).
- It will buy you time to enlist the help of others with more experience --your fellow graduate students, your CL friend that programs, more advanced graduate students.

Why it matters to think about how we manage our data?

- Start to develop these skills NOW.
  - It's easier when projects are smaller and simpler (i.e. your term papers!)
  - FYI... Your directories with sound files and TextGrids will get huge as you make progress in your degree.

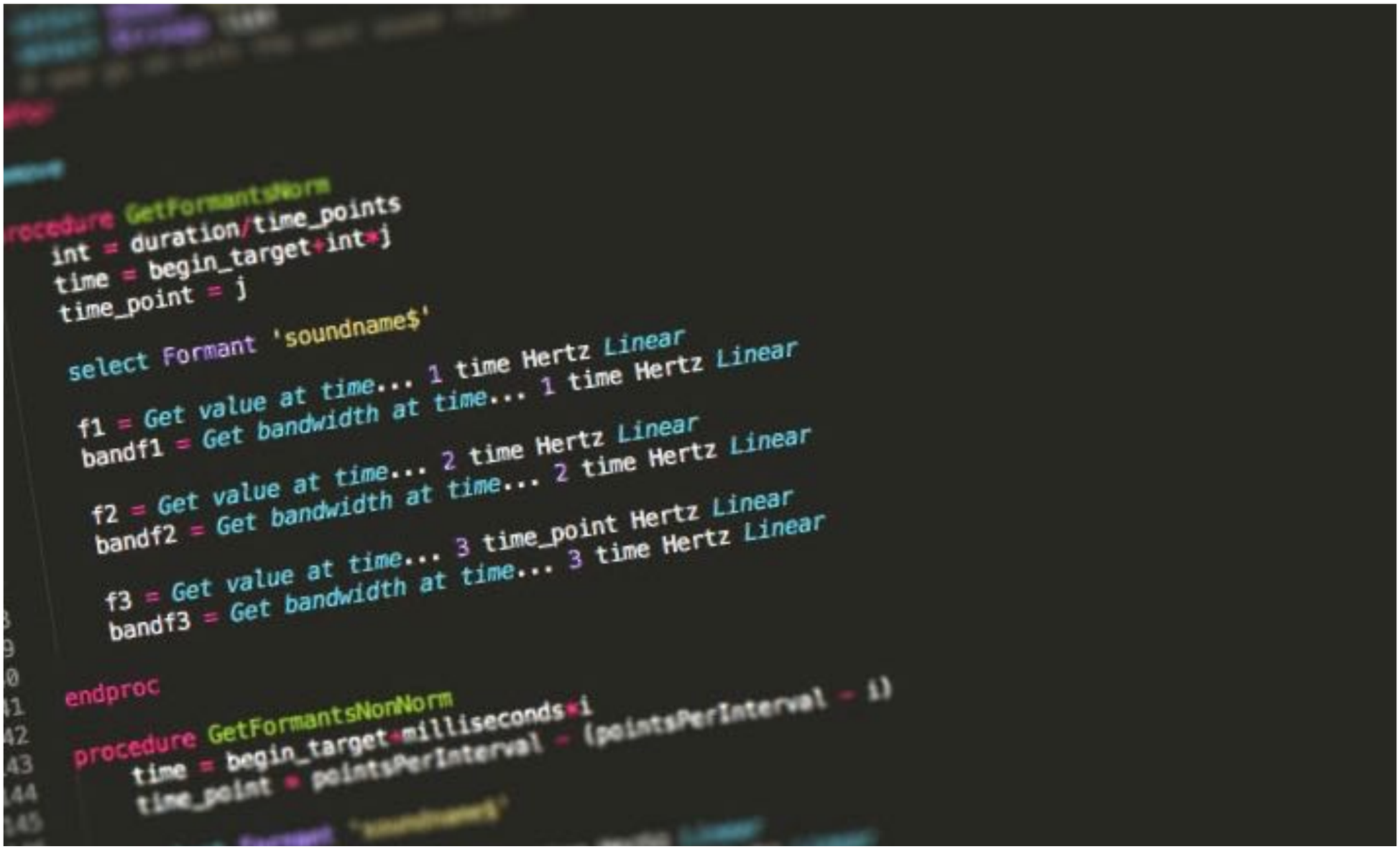
Things Praat scripts do

- Automatize repetitive routines.
- Makes it easier for us to perform complex operations.
- Do one (or two, or three, …) task the exact same way every time.
- Provides a way to document our workflow.
- Remind us that we are wrong.

Things scripts DO NOT do

- Replace us.
- Think for us.
- Make decisions for us.

Questions?



Let’s talk scripting

Before we begin

- You should already have Praat
- Please launch it and check your version number (Praat —> About Praat…)
- Then check the Praat home page for the current latest version and see what's new

Note:

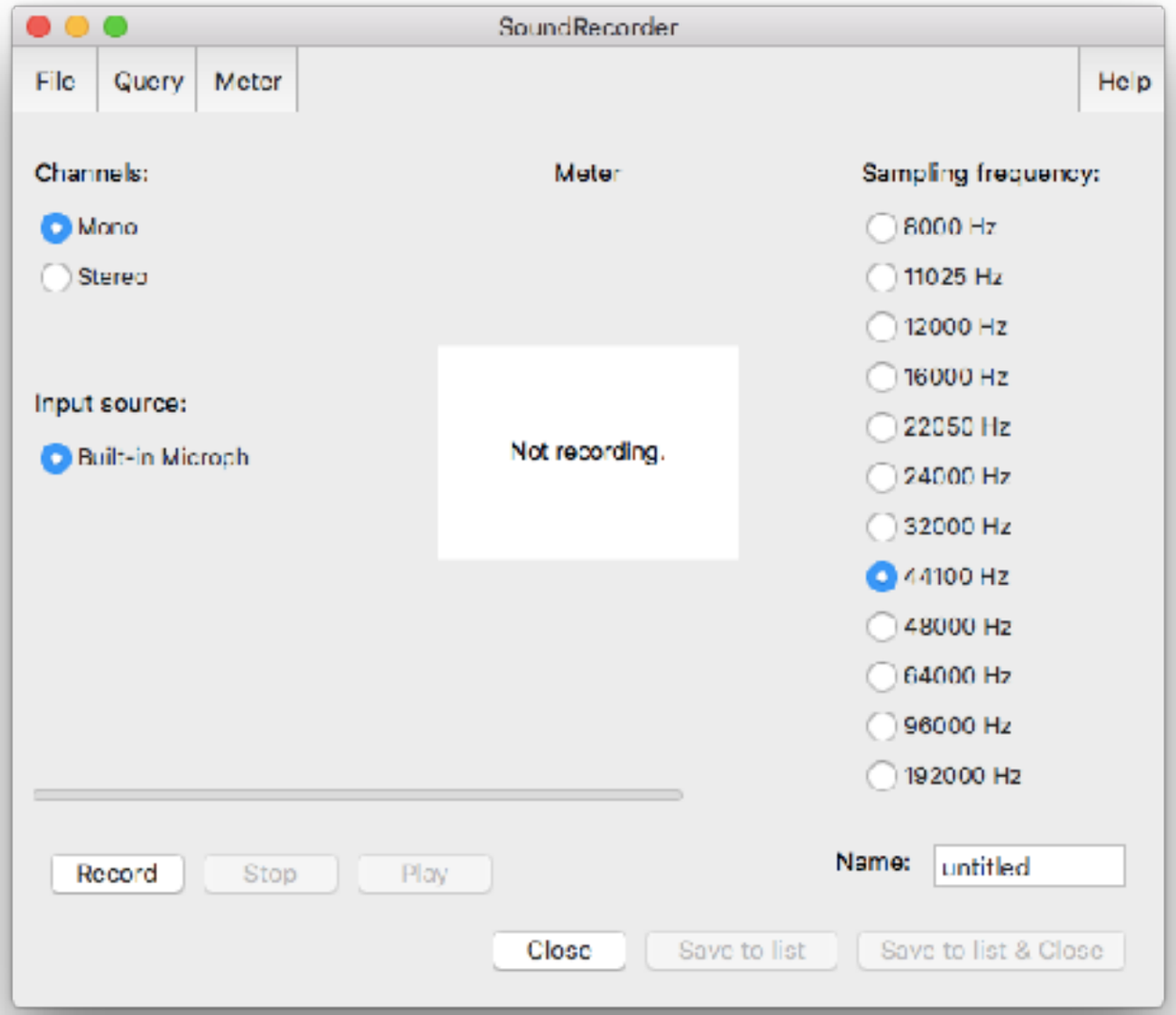
It is standard practice to include version number and download date when citing Praat in published work. Be sure to make a note of these when upgrading

Programming vs Scripting?

- There is a real difference between programming and scripting, but that difference is not important here.
- **Programming** refers to writing stand-alone software that runs independent of an external program.
- **Scripting** refers to writing commands to control another program and manipulate its data files.
- What they have in common is the need to define your inputs and outputs clearly, translate your desires into a language the computer can understand, and debug that language. In this class we will apply the core principles of computer programming to the needs of linguists working with Praat.

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”



ScriptEditor (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.

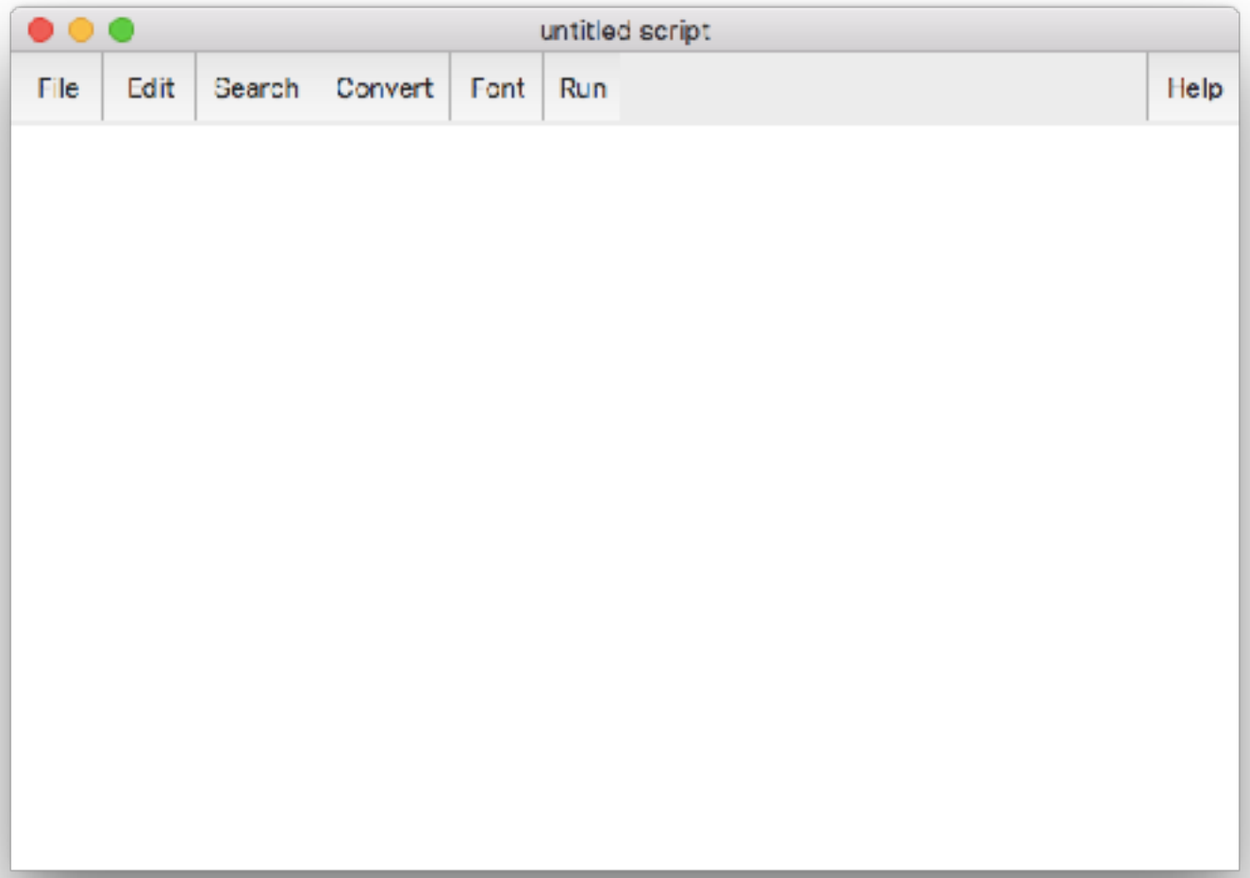
1. The History Mechanism

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?

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

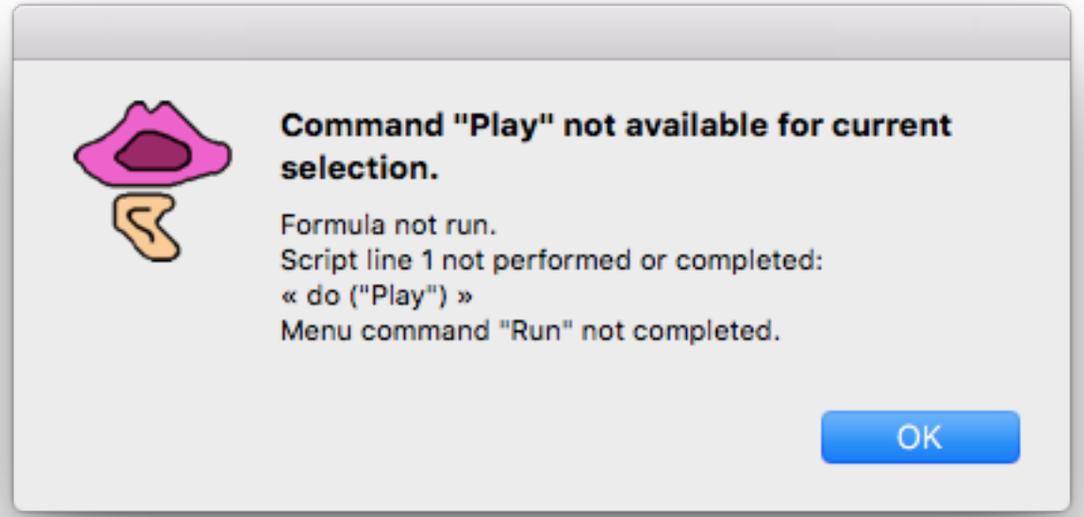
1. Erase any text in the ScriptEditor window (select all and then Edit → Erase)
2. Select the Sound object in the Objects window
3. In the ScriptEditor, type the Praat script:

```
do ("Play")
```

4. Run your new script (Run → Run) or (⌘R)
5. Did it play without error?
6. Now, create a new TextGrid object (Objects window → Annotate → To TextGrid...) & accept defaults by clicking [ok]
7. What happens when you run the same script now?

Read the error message!

- Praat's error messages are pretty informative and often tell you precisely what is wrong with your script.
- In this case, you should have gotten something like:



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.

- In a script you have two choices...

Selecting an object

- Selecting an object makes it active for Praat.
- You might think of it as focusing Praat's attention on the object or making it the focus of any commands or requests for data.
- 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, e.g.:

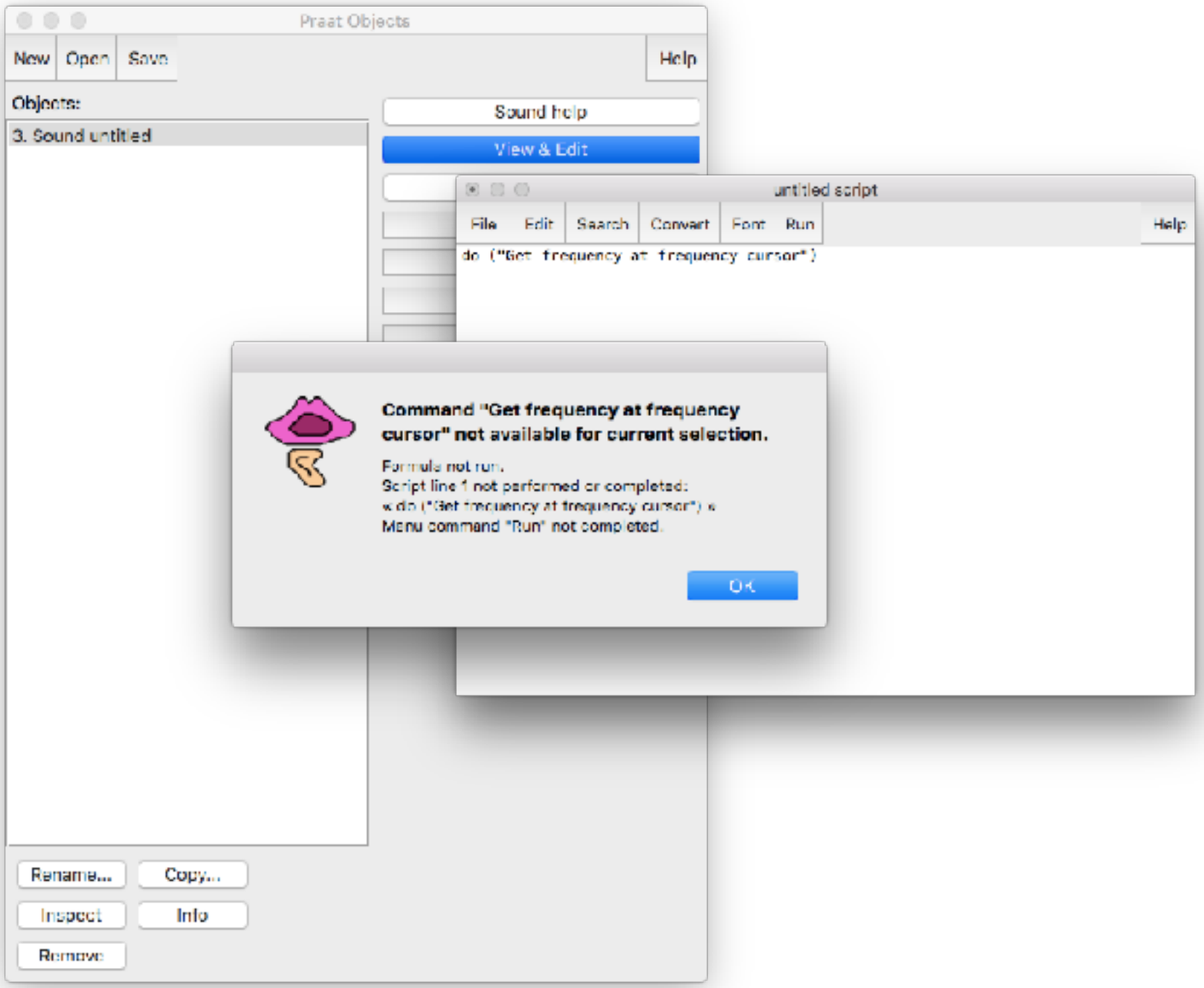
```
selectObject( "Sound untitled" )

selectObject( "TextGrid untitled" )
```

- More generally: `selectObject( "ObjectType objectname" )` (manual)

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 10 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).
- I'll show you what the error looks like...

SoundEditor command from Objects window



Exercise: Scripting for Humans

- What is a task that you would want to do in Praat?
- Write down a step-by-step description of what you do to go from sound file to usable data

Managing scripts

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

Files

- Rarely you might write Praat scripts that you only need or want for the current session
- But *typically* you will want to save your script to reuse later
- 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.
- Be sure to use comments!

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
      do("Play")
- And while you can't use # for an **inline comment:**

  *do("Play") # user needs to hear, and approve, the sound
- You *can* do this:

  do("Play") ; user needs to hear, and approve, the sound

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

do ( "Erase all" )
do ( "Text top...", "yes", "hello, world!" )

# question:
# what are the ellipses 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

```
# store the greeting in a string variable
# (more on these later)

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$, "!" )
```

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$, "!" )
appendInfoLine ( "hello, " + greeting$ + "!" )
```

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$'!" )
```

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 the bucket's name.

• 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!"

Variables (cont.)

- Variables have three attributes:

1. **name:** a unique identifier for the memory address
2. **type:** defines the affordances and behaviors of the variable
3. **value:** the information stored in the variable –can be a **literal** or **constant** value typed-into the script, read from the world, or calculated at runtime

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 )
```