Praat Scripting Workshop

Jonathan Havenhill

Graduate Linguistics Student Association
Georgetown University
jeh241@georgetown.edu

November 15, 2014

Files for today

- You should download the following files:
  - `github.com/jhavenhill/praatscriptingworkshop`
- If you don't have Praat installed, you should download it now:
  - `www.fon.hum.uva.nl/praat/`

Some basic information

What are the benefits of scripting Praat?[1]

- Save time:
  - Scripting takes a while, but not as long as, say, measuring 800 tokens for 20 variables each
- Consistency:
  - Doing things by hand can lead to a lot of mistakes
- Some things are too difficult or annoying to do by hand:
  - Calculating vowel midpoints
  - Measuring formants at F1 maximum
- It forces you to think about what exactly you're measuring

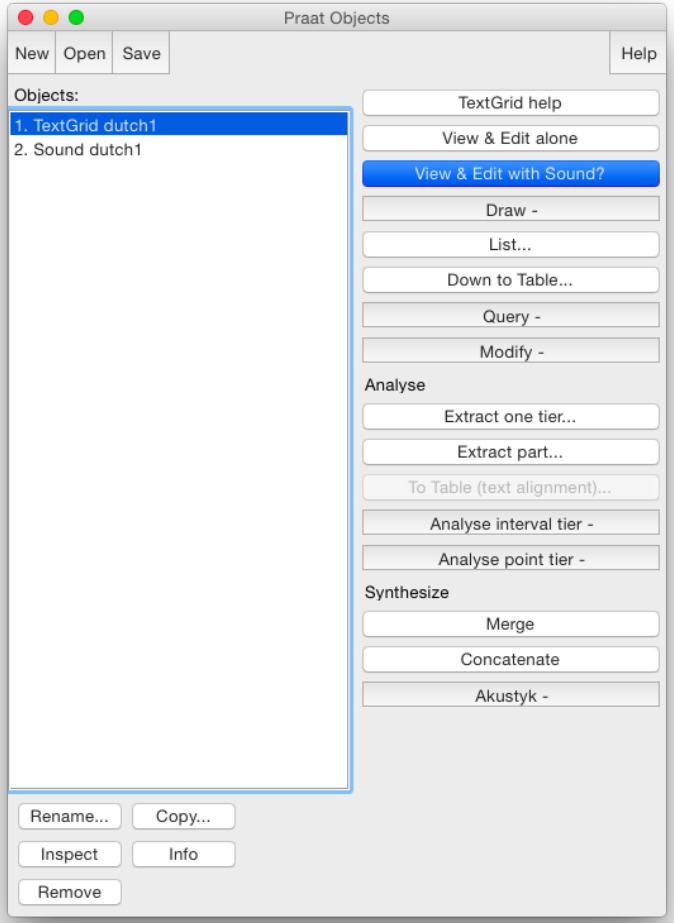
[1] Adapted from Shigeto Kawahara

Demo: Consistency

- Open "dutch1.wav"
- Get the values of F1 and F2 at the midpoint
  - What did you get?
- Scripts require you to be explicit:

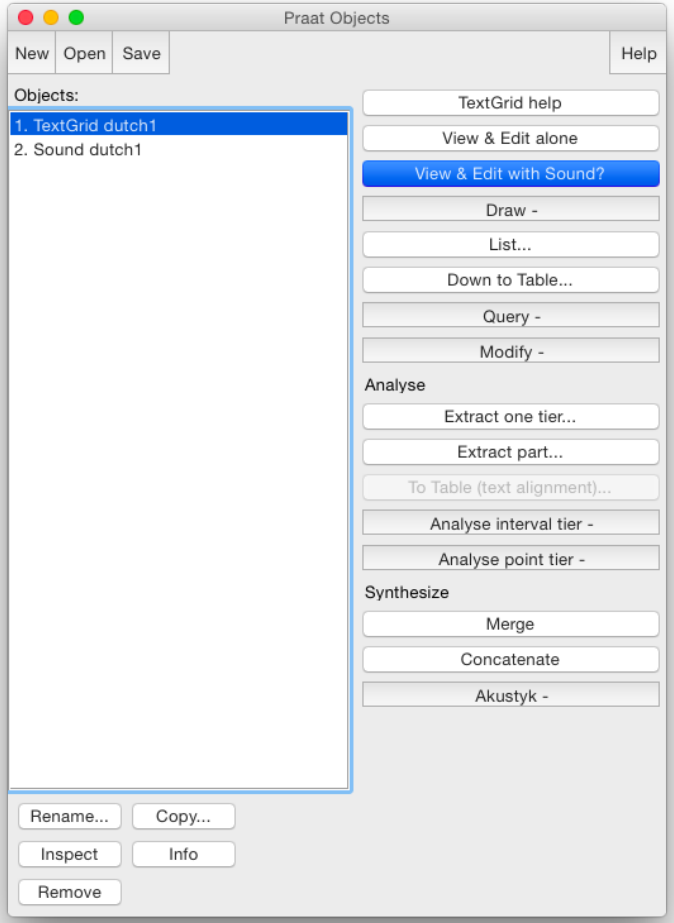
```
selectObject ("Sound dutch1")
do ("To Formant (burg)...", 0, 5, 5500,
 0.025, 50)
do ("Get value at time...", 1, 0.382,
 "Hertz", "Linear")
```

Objects Window



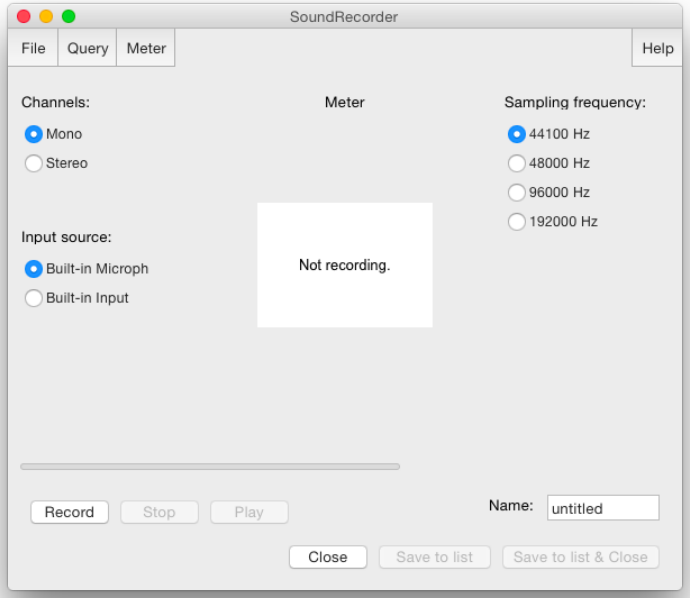
- This is the main window in Praat
- Every file that you open or create with Praat will appear here
- To apply a script to an Object, you must open it first

Objects Window



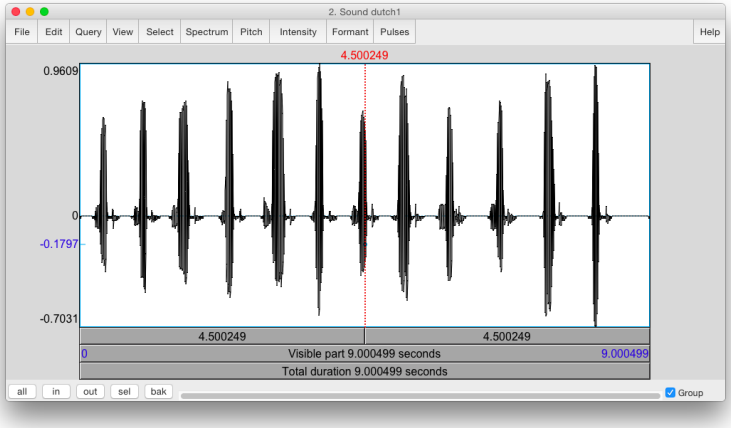
- There are many types of Object, e.g. Sound and TextGrid
- Each has different actions you can apply to it
  - These appear as dynamic buttons on the right side
- The buttons at the bottom are fixed

SoundRecorder



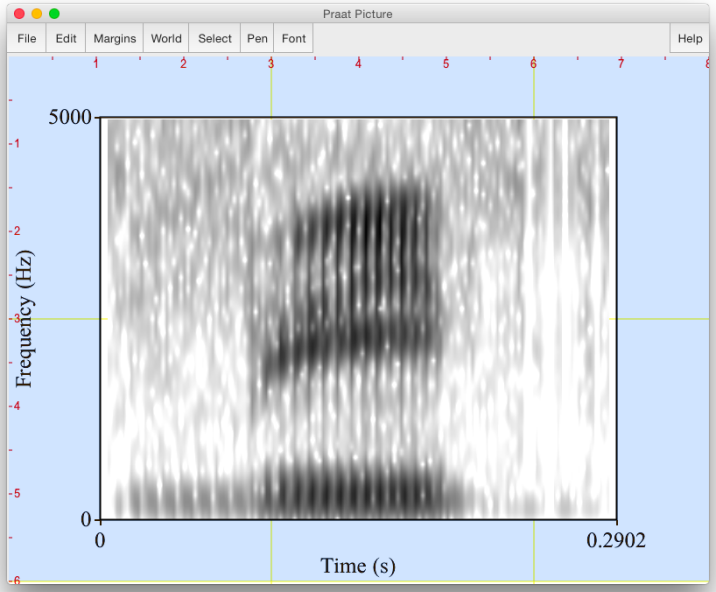
- New > Record mono sound . . .
- Record yourself saying **"Hello, World"** in the language of your choice
- Select Save to list & Close
- This saves your sound to the Objects window (but does not save it to disk)

Editor windows



- Some object types can be opened in an Editor window
- Each has its own, unique Editor window (there are 13 in total)
- You can script Editor functions, but this is generally dispreferred

Picture window

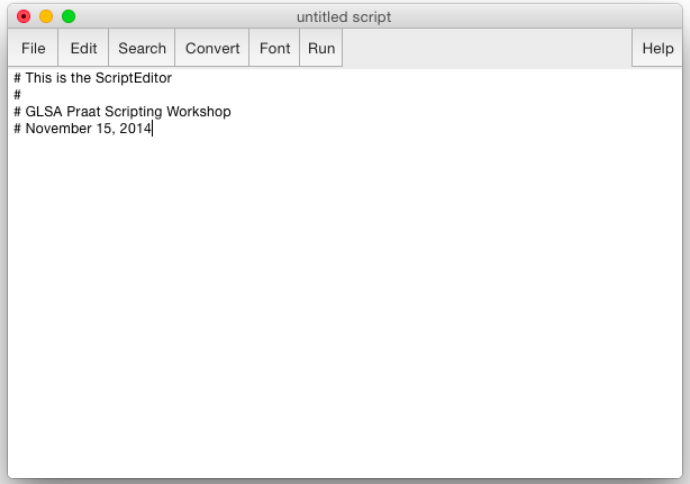


- This window is actually useful
- You can create high-quality graphs and images for using in your papers (much better than screenshots of the Editor window)

Picture window demo

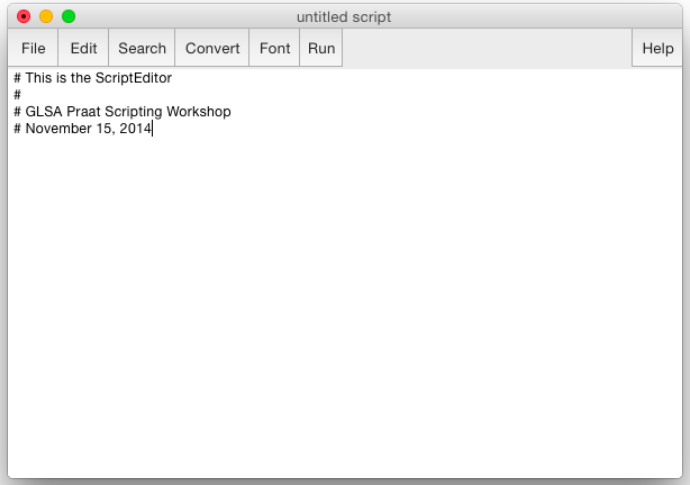
- First, draw a Spectrogram of the sound you recorded:
  - Select Sound in Objects window
  - Analyze Spectrum > To Spectrogram... > OK
  - Select Spectrogram in Objects window
  - Draw > Paint > OK
- Then, try playing around with some of the functions in the "World" menu (use ⌘/Ctrl + E to start over)

Finally: the ScriptEditor



- Praat > New Praat script
- Once you start writing complex scripts, you'll probably want to use an editor made for coding
- But you'll always need the ScriptEditor for two important features.

ScriptEditor



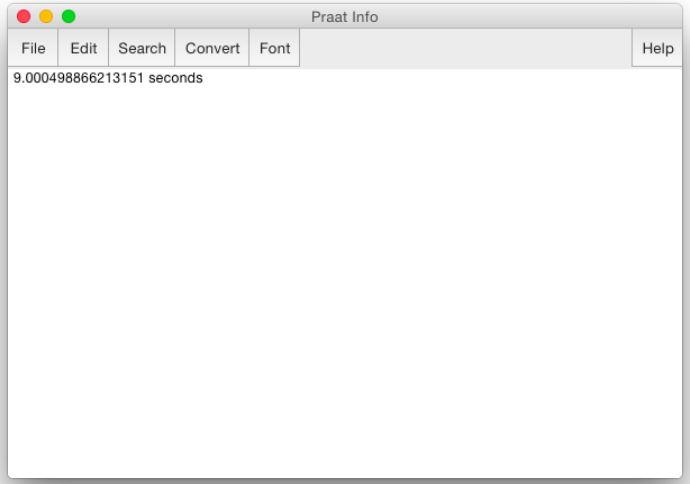
- Run and Run Selection
  - Run: ⌘/Ctrl + R
  - Run selection: ⌘/Ctrl + T
- Command History
  - Edit > Paste history
  - ⌘/Ctrl + H

Exercise

- First, save the "Hello World" sound you recorded to somewhere on your computer
- Then, write a script that does the following:
  - Read the .wav from disk
  - Paint a spectrogram of the sound in the Picture window, showing 0-8000 Hz.

  Bonus:
  - Give the spectrogram dimensions of 4 in x 6 in (hint: size is determined by the viewport)
  - Give your Spectrogram a title like "Hello World" (hint: you want the title to go in the margin)
  - Save your Spectrogram as a PDF

One more: the Info window



- The Info window displays...info!
- This is the window that pops up when you take formant measurements
- But you can write things to it as well

Info window

- Type the following into the ScriptEditor:

```
writeInfoLine ("Hello, world!")
```

- Click Run > Run
- Now, try adding another line:

```
writeInfoLine ("Hello, world!")
writeInfoLine ("How art thou?")
```

- What happens?

Info window

- We need another command:

```
writeInfoLine ("Hello, world!")
appendInfoLine ("How art thou?")
```

- `writeInfoLine` erases the contents of the Info window before printing, `appendInfoLine` doesn't

Comments

- It's important to comment your code, so others (and Future You) will know how it works
- Comments in Praat start with # or ; (semicolon):

```
# This is a comment
; This is also a comment
```

- Comments starting with # can be used inline, while comments starting with ; cannot
- It's also a good idea to leave a blurb at the beginning of your script that explains what it does and who wrote it:

```
# This script extracts formants
# and vowel duration and saves
# the values to a .csv
#
# Jon Havenhill - 15 Nov 2014
```

White Space

- Praat doesn't care about white space, unlike some other languages (such as Python)
- However, it is good practice to indent your code:

```
for i from 1 to 10
    do (...)
    for j from 2 to 20
        do (...)
    endfor
endfor
```

- Four spaces for each level is customary
- Praat also ignores blank lines, so you can use them to visually organize your code

Long lines

- Sometimes lines get really long:

```
appendInfoLine("analyzeFilesResults.txt", "dialec
tab$, "subject", tab$, "real", tab$,
"consonant", tab$, "following vowel",
tab$, "notes", tab$, "repetition",
tab$, "word", tab$, "environment", tab$,
"dur preceding V", tab$, "dur preceding N",
tab$, "closure dur", tab$, "voicing dur",
tab$, "vowel intensity", tab$, "VOT", tab$, "frica
"F1 into closure", tab$, "F0 out of closure",
tab$, "F1 out of closure")
```

- So you can break them up with an ellipsis (...)

```
Create Sound from formula: "windowedSine", 1, 0,
... "0.5 * sin(2*pi*1000*x) * exp(-0.5*((x-0.5)/0
```

Variables

- Variables allow you to store some piece of information in memory for later use
- Praat has two kinds:
  - Numeric variables
  - String variables
- Variable names must start with a lower-case letter, but they may contain capital letters, numbers, or underscores:

`formant`	not: `Formant`
`formant1`	not: `formant!`
`f1`	not: `F1`
`f1_max`	not: `f1-max`

Numeric Variables

- Numeric variables may contain integers or decimal (real) numbers
- Values less than 1 must have a preceding 0
  - `0.5` *not* `.5`
- Variables are assigned using the equal sign:

```
formant = 1500
writeInfoLine ("The formant value is: ",
... formant, ".")
```

- What will this give us?

```
formant = 1500
formant = 1650
writeInfoLine ("The formant value is: ",
... formant, ".")
```

Numeric Variables

- Variables may also contain other variables:

```
formant1 = 500
formant2 = formant1 * 2
writeInfoLine ("F1 is half of: ",
... formant2, ".")
```

- What will this give us?

```
b = 3.1
c = b * 2
b = 5.8
writeInfoLine ("The value of c is ", c, ".")
```

- It gives us **6.2**, not **11.6**. `c` is assigned the value of `b` at the moment of evaluation

Numeric Operators

- Math:
  - Negation: `-`
  - Exponentiation: `^`
  - Multiplication: `*`
  - Division: `/`
  - Addition: `+`
  - Subtraction: `-`
- Integer division (same precedence as `*` and `/`):
  - `div`: division rounded down
  - `mod`: the remainder
- What do the following operations give us?

```
f0 = 188
a = f0 div 10
b = f0 mod 10
writeInfoLine ("The value of a is ", a, ".")
appendInfoLine ("The value of b is ", b, ".")
```

Numeric Variables

- Constants:
  - `pi` (π)
  - `e` (e)
  - (You can't use these as names of variables)
- Functions:
  - Convert to string (and back):

```
f0$ = string$ (f0)
f0 = number (f0$)
```

  - Round to *n* decimal places:

```
f0 = 233.23792719
writeInfoLine ("The F0 is: ", fixed$(f0, 2),
... " Hz.")
```

  - Display as percentage with *n* decimal places:

```
score = 87.3 / 100
writeInfoLine ("Your score is: ",
... percent$(score, 2))
```

String Variables

- String variables must end with a dollar sign ('$')

```
helloworld$ = "Hello, World!"
```

- When assigning a value to a string variable, the string must be enclosed in double quotation marks
- Can be empty: `empty$ = ""`

Concatenation/Truncation

- Like numeric variables, string variables have operators:
  - Concatenate: `a$ + b$`
  - Truncate: `a$ - b$`

```
fileName$ = "vowels.wav"
tgName$ = fileName$ - ".wav" + ".TextGrid"
writeInfoLine("The name of the TextGrid
... is: ", tgName$)
```

- What happens if the file is a `.aiff`?

```
fileName$ = "vowels.aiff"
tgName$ = fileName$ - ".wav" + ".TextGrid"
writeInfoLine("The name of the TextGrid
... is: ", tgName$)
```

String Functions

- There are also a number of functions that can be used with strings
- Some return strings, others numbers
  - `length (string$)`
  - `number (string$)`

```
word$ = "exquisite"
wordLength = length (word$)
writeInfoLine("There are ", wordLength,
... " characters in ", word$)
```

```
string$ = "5e6"
value = number (string$) * 2
writeInfoLine("The value is: ", value)
```

String Functions

- Extracting parts of strings:
  - `left$ (string$, n)`
  - `right$ (string$, n)`
  - `mid$ (string$, n, m)`

```
word$ = "working"
wordEnding$ = right$ (word$, 3)
wordStem$ = left$ (word$, 4)
writeInfoLine("The suffix is: ",
... wordEnding$)
appendInfoLine("The stem is: ", wordStem$)
```

```
word$ = "working"
preceding$ = mid$ (word$, 4, 1)
writeInfoLine("The preceding segment is: ",
... preceding$)
```

String Functions

- Extracting parts of strings:
  - `index (stringA$, stringB$)`:
    Finds first occurrence of *stringB$* in *stringA$*
  - `rindex (stringA$, stringB$)`:
    Finds last occurrence

```
word$ = "working"
vowel$ = "i"
first = index (word$, vowel$)
writeInfoLine("The first occurrence of ""i""
... is the ", first, "th character")
```

- NB: the index starts at 1, not at 0 like in some other languages

String Functions

- Searching strings:
  - `startsWith (stringA$, stringB$)`
  - `endsWith (stringA$, stringB$)`

  Retuns 1 if true, 0 if false

```
word$ = "working"
suffix$ = "ing"
value = endsWith (word$, suffix$)
writeInfoLine("This word ends in ""ing"": ",
... value)
```

String Functions

- Replace:
  - `replace$ (stringA$, stringB$, stringC$, n)`
- Returns a string *like* `stringA$`, but where `stringB$` has been replaced with `stringC$` *n* times.

```
word$ = "working"
suffix$ = "ing"
newSuffix$ = "ed"
newWord$ = replace$ (word$, suffix$,
... newSuffix$, 1)
writeInfoLine("The new word is: ",
... newWord$)
```

Predefined Variables

- `praatVersion`: Current version of Praat (e.g. 5401)
- `macintosh`/`windows`/`unix`: value of 1 if script is running on that platform
- `defaultDirectory$`: Directory in which the script is saved
- `newline$`: Inserts a linebreak
- `tab$`: Inserts a tab

```
f1 = 650
f2 = 1500
newSuffix$ = "ed"
writeInfoLine("F1 is ", f1, " Hz.",
... newline$, "F2 is", f2, " Hz.")
```

Operators

- In addition to mathematical operations, there are comparison operators:
  - = equal
  - <> does not equal
  - < less than
  - > greater than
  - <= less than or equal
  - >= greater than or equal
- You'll need these for conditionals and `while` loops
- Some comparison operators from other languages can be used, for example == for equal and `!=` for unequal
- Comparison operations return a value of 0 (if false) or 1 (if true)—these values can be saved to a variable

Operators

- Comparison operators can also be used with string variables:
  - `a$ = b$`: true if strings are equal
  - `a$ <> b$`: true if strings are unequal
- These are a bit less intuitive:
  - `a$ < b$`: true if `a$` precedes `b$` in ASCII sorting order
  - `a$ > b$`: true if `b$` precedes `a$`
  - `a$ <= b$`: true if `a$` precedes or is equal to `b$`
  - `a$ >= b$`: true if `b$` precedes or is equal to `a$`
- ASCII sorting order:
  - Numbers precede letters
  - Capital letters precede lower case letters
  - Numbers are sorted by their individual characters, e.g., 10 comes before 2

Conditionals

- `if`: introduces conditional
- `elsif`: introduces possible alternate outcome
- `else`: executed if none of the preceding tests were true
- `endif`: ends conditional

```
if x = y
    do something
elsif x = z
    do something else
else
    do yet another thing
endif
```

- NB: It's much easier to read if you indent, but it's not strictly necessary. But use spaces, not tabs. Tabs are rendered inconsistently across text editors.

Conditionals

- Logical operators:
  - not
  - and
  - or

```
if ((x = y) and (x = w))
    do something
elsif ((x = z) or (x = q))
    do something else
elsif ((x = s) and not (x = t))
    do something else
else
    do yet another thing
endif
```

- `not` takes precedence over `and`, which takes precedence over `or`. But it's easier if you just use parentheses.

Conditionals

- Let's write one together
  - Generate a random number from 1 to 100
  - Test whether number is greater than 50
  - Write the results to the Info line
-

```
number = randomInteger(1, 100)

if number > 50
    writeInfoLine ("The number was
    ... greater than 50.")
elsif number = 50
    writeInfoLine ("The number was 50.")
else
    writeInfoLine ("The number was
    ... less than 50.")
endif
```

Exercise: Variables and Conditionals

- Write a script to determine whether a given year is a leap year
- Leap years occur every 4 years, unless the year is divisible by 100. Years divisible by 400 are leap years as well.
- Write the results to the Info window
- Hint: If a year is divisible by 4, its remainder will be 0
- Bonus:
  - Prompt the user for the year (see `http://www.fon.hum.uva.nl/praat/manual/Scripting_6_1__Arguments_to_the_script.html`)
  - Prompt the user for a date in some format, like 11/15/2014 or November 2014, then determine whether that date is in a leap year

Exercise: Basic Solution

```
year = 2004

if (((year mod 4 == 0) and (year mod 100 <> 0))
... or (year mod 400 == 0))
    writeInfoLine (year, "is a leap year!")
else
    writeInfoLine (year, "is not a leap year!")
endif
```

Exercise: Bonus Solution

```
form Give me a year:
    integer year
endform

if (((year mod 4 == 0) and (year mod 100 <> 0))
... or (year mod 400 == 0))
    writeInfoLine (year, "is a leap year!")
else
    writeInfoLine (year, "is not a leap year!")
endif
```

Exercise: Bonus Solution 2

```
form Give me a date:
    text date
endform
index = index_regex(date$, "[0-9][0-9][0-9][0-9]")
if index <> 0
    year = number (mid$ (date$, index, 4))
    if (((year mod 4 == 0) and (year mod 100 <> 0))
    ... or (year mod 400 == 0))
        writeInfoLine (year, " is a leap year!")
    else
        writeInfoLine (year, " is not a leap year!")
    endif
else
    writeInfoLine ("Please enter a date with
    ... a 4-digit year.")
endif
```

Review

Loops

- The best reason to script Praat—taking care of mundane, repetitive tasks
- A loop is a set of commands that is repeated until some condition is met
- There are three basic types of loop in Praat:
  - For loop
  - While loop
  - Until loop

For Loops

- A for loop is a loop which repeats a given number of times
- A skeleton for loop:

```
for i from x to y
    do something
endfor
```

- The statement between `for` and `endfor` is executed, and `i` increases by 1 each time
- The `from x` statement is optional. If you don't include it, `i` starts at 1

For Loops

- Plot points to the picture window

```
do ("Erase all")
do ("Axes...", 0, 100, 0, 100)
m = 2
b = 3
for x from 1 to 100
    y = m * x + b
    do ("Draw circle...", x, y, 1)
endfor
```

Exercise: For Loops

- Let's make a loop that simulates a coin toss
- Use a for loop to:
  - Generate a random integer "between" 1 and 2
  - Use an if statement to save the result as either a heads or tails
  - Repeat 100 times
  - Write the results to the Info window

Solution: Coin Toss

```
heads = 0
tails = 0
for toss from 1 to 100
    flip = randomInteger(0, 1)
    if flip = 0
        heads = heads + 1
    else
        tails = tails + 1
    endif
endfor
writeInfoLine ("Heads: ", heads,
... newline$, "Tails: ", tails)
```

Until Loops

- An until loop is one which repeats an infinite number of times until some condition is met
- A skeleton until loop:

```
repeat
    do something
until x = y
```

- The statement will always be completed at least once.
  - Try the following. Clear your Info window first with File > Clear.

```
x = 7
repeat
    appendInfoLine ("test")
until x > 1
```

- The reason why is intuitive: the test isn't conducted until after the first iteration.

Until Loops

- Let's write an until loop
- This script will measure the number of tries it takes to roll 12 with two dice.

```
throws = 0
repeat
    roll = randomInteger (1,6)
    ... + randomInteger (1,6)
    throws = throws + 1
until roll = 12
writeInfoLine ("It took ", throws, " throws
... to roll 12 with two dice.")
```

While Loops

- A while loop repeats until some condition no longer holds true
- A skeleton while loop:

```
while x = y
    do something
endwhile
```

- Unlike until loops, while loops may execute zero times.
  - Let's return to the previous example. Again, clear your Info window first using File > Clear.

```
x = 7
while x < 1
    appendInfoLine ("test")
endwhile
```

- Nothing happens. What would happen if the test was `while x > 1`?

While Loops

- Let's convert our dice roll script to use a while loop. What do we need to change?
-
```
throws = 0
roll = 0
while roll <> 12
    roll = randomInteger (1,6)
    ... + randomInteger (1,6)
    throws = throws + 1
endwhile
writeInfoLine ("It took ", throws, " throws
... to roll 12 with two dice.")
```

While Loops vs. Until Loops

- While loops and until loops are very similar
- Choosing one is really a matter of preference, and whichever is best suited for a specific application

Exercise: Nested loops

- Let's write a script to find all the prime numbers less than n
  - Use a loop to iterate through each number x to n
  - Use a nested loop to iterate through each possible divisor (y) of x
  - Test whether x is evenly divisible by y
  - If y is prime, print it to the Info window

Solution: Nested loops

```
max = 100
x = 1
while x < max
x += 1
not_prime = 0
for y from 2 to x-1
    if x mod y = 0
        not_prime += 1
    endif
endfor
if not_prime = 0
    appendInfoLine(x, " is a prime number!")
endif
endwhile
```

TextGrids

- What is a TextGrid, anyway?
- It's really just a text file:

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

xmin = 0
xmax = 79.79486052387188
tiers? <exists>
size = 4
item []:
    item [1]:
        class = "IntervalTier"
        name = "Phrase"
        xmin = 0
        xmax = 79.79486052387188
        intervals: size = 41
        intervals [1]:
            xmin = 0
            xmax = 0.7365941687547206
            text = ""
        intervals [2]:
            xmin = 0.7365941687547206
            xmax = 2.2336095443531008
            text = "say feet again"
```

TextGrids

- What is a TextGrid, anyway?
- It's really just a text file.
- It contains attributes you can extract, either through the dynamic buttons in the Object window, or through a script.
- These attributes include:
  - A start time
  - A stop time
  - Tiers
- A tier can be an IntervalTier, which contains a set of intervals, each with its own start and stop time
- Or it can be a point tier, just a labeled point in time

TextGrids

- TextGrids are perhaps the most useful thing you can script
- You can easily automate extracting any measurement you would take by hand
- And some measurements that are tedious to take by hand, e.g., measuring formants at the point of F1 maximum

TextGrids

- Let's create a TextGrid, first by hand, then using a script.
- Open the "Hello World" sound you recorded earlier, or create a new one.
- Create a new TextGrid with Annotate > To TextGrid...
- Name the tiers: `words`, `vowels`, and `midpoints`. Make `midpoints` be a point tier
- Now, open the ScriptEditor and paste your command history

```
selectObject ("Sound helloworld")
do ("To TextGrid...", "words vowels
... midpoints", "midpoints")
```

- Tier names are given in a space-delimited string of names

Practice: Working with TextGrids

- Annotate your "Hello, World" file
- Start with the words tier. Click the point where the word starts, and choose Boundary > Add on tier 1
- Do the same at the point where the word ends.
- Click between the boundaries and type a label
- Repeat for each word and vowel
- Save as a text file File > Save TextGrid as text file... or ⌘/Ctrl + S

Practice: Working with TextGrids

- Okay, now let's write a basic script to extract some measurements
- Get the duration of each vowel
- Add a point at the midpoint to the midpoint tier
- Use the buttons (in the Objects window) to get the commands, then convert it to a loop
- Hints:
  - Your commands will query the TextGrid, not the Sound!
  - Use an if statement to determine whether an interval is labeled (i.e. is a vowel)

Solution: TG Script

```
selectObject: "TextGrid HelloWorld"
nInt = Get number of intervals: 2

for i to nInt
    label$ = Get label of interval: 2, i

    if label$ <> ""
        intStart = Get start point: 2, i
        intEnd = Get end point: 2, i

        midpoint = (intEnd + intStart) / 2
        Insert point: 3, midpoint, ""
```

Solution: TG Script (cont.)

```
        duration = intEnd - intStart
        appendInfoLine: "The duration of """,
        ... label$, """ is: ",
        ... fixed$ (duration, 2), " seconds."
    endif
endfor
```

Writing to external files

- It's pretty simple: instead of `writeInfoLine`, we use `writeFileLine`

```
writeFileLine ("FileName.txt", ...)
```

- Note that the file name is a string which precedes what you want to export
- Like `writeInfoLine`, `writeFileLine` overwrites the existing contents
- So for subsequent lines, you'll want to use `appendFileLine`

Writing to external files

- A useful function: `fileReadable(...)`
- This function checks whether a file exists and returns 1 if true
- It can be used in a conditional like so:

```
if fileReadable ("analysisLog.txt") == 1
    writeInfoLine("analysisLog.txt detected")
    appendFileLine("analysisLog.txt", newline$)
else
    writeInfoLine("File not detected. Creating
    ... analyzeFiles.log...")
    writeFileLine("analysisLog.txt", newline$)
endif
```

Writing to external files

- Praat's default directory is the directory where the script is saved.
- It's best to use relative paths to files.
- If you want to access "/Volumes/User/praat/stimuli/sound.wav", use:
  - "sound.wav" if the script is in the folder "stimuli"
  - "stimuli/sound.wav" if the script is in "praat"
  - "../stimuli/sound.wav" if the script is in "scripts", which is in "praat"

Exercise: TextGrids 2

- Open the file "sayXagain.wav" and its associated TextGrid
- Write a script to:
  - Measure the F1 and F2 of each vowel
  - Measure the duration of each vowel
  - Print the values to the Info window
- Bonus:
  - Write the values to a tab-delimited text file with the header:
    word vowel duration F1 F2
  - Plot the vowels to the Picture window (and save as a PDF!)

Resources

- When in doubt, check the manual:
  - `http://www.fon.hum.uva.nl/praat/manual/Scripting.html`
  - Also available under Help
- Some helpful guides are found in the "resources" folder
- The Internet: many of the scripts you will want to write have already been written, you'll just have to adapt them to your needs