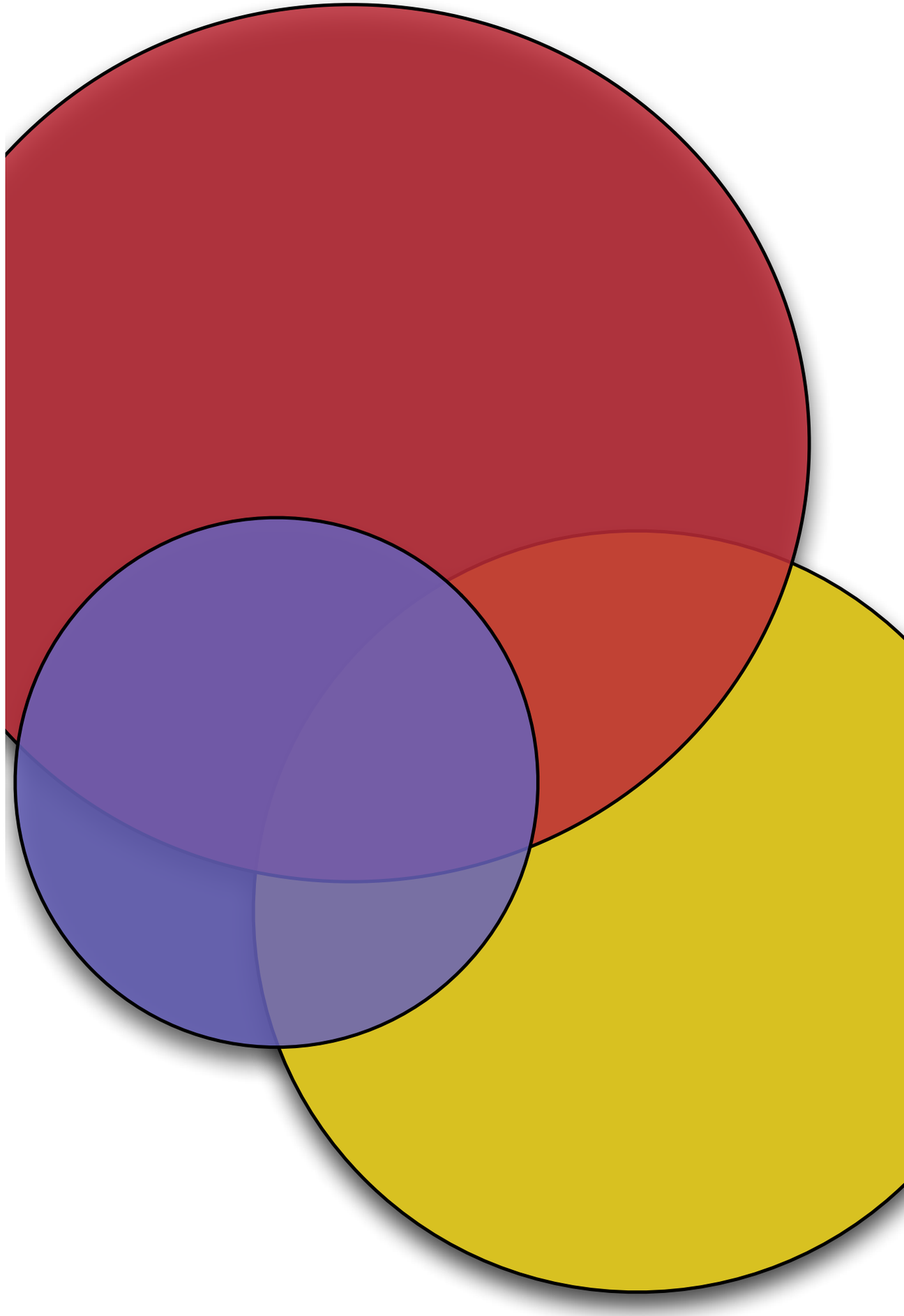
Praat Scripting

Goodies

Today

- TextGrid looping solution
- Input at runtime: form & pause
- Procedures
- Other people's code
    - Scripting the Editors
    - Duration logger
    - Resources
- Next Steps



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 day3Code.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.

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 day3Code.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 day3Code.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

Procedures

Procedures

- We know that we can use loops when we want Praat to repeat the same code multiple times

- Loops make sense when the block of code needs to repeat many times sequentially

- What if you have a block of code you want to run multiple times but not all at once?

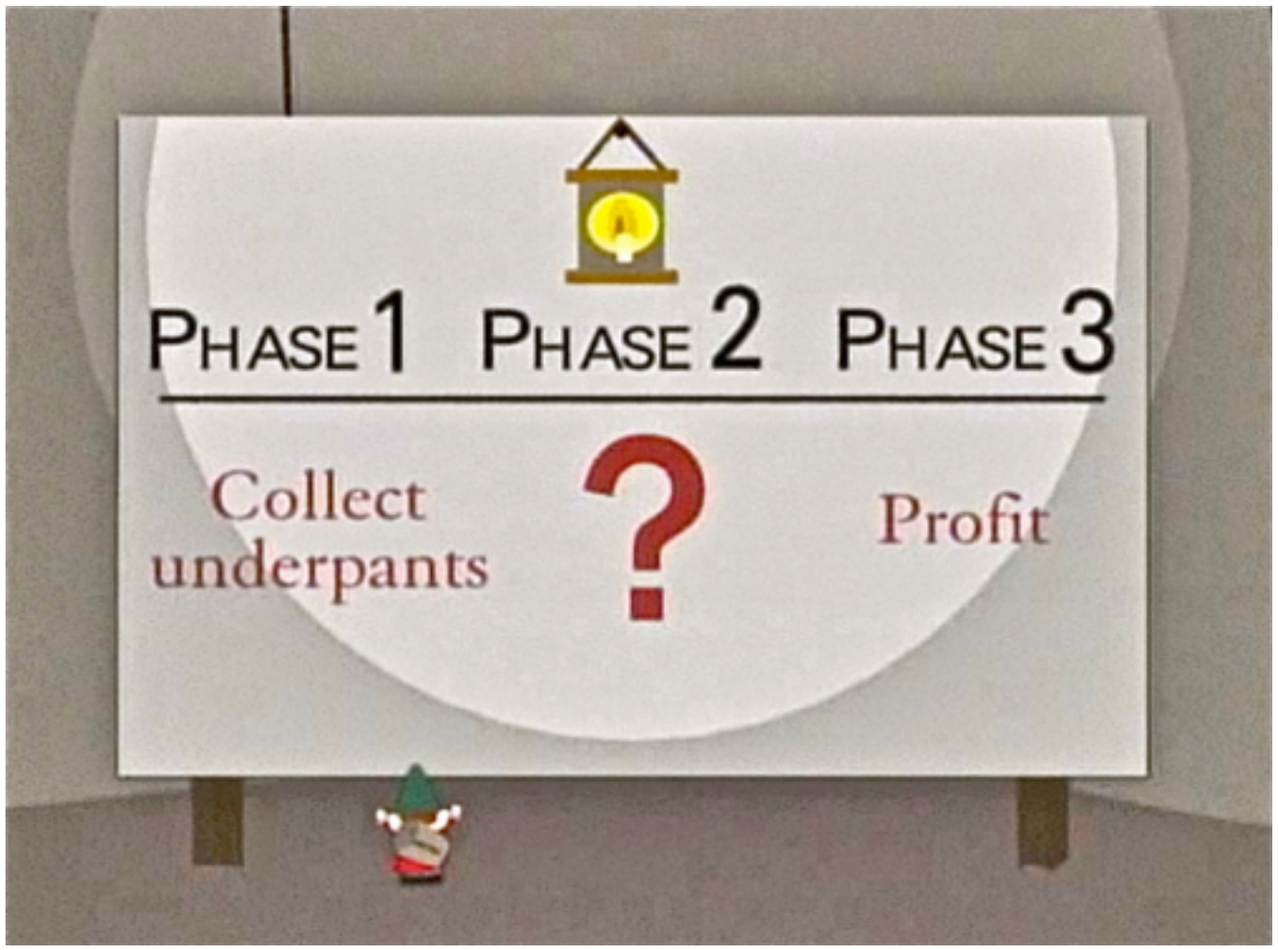
Procedures in real life

- Imagine you are making a To Do list and it includes items like:
    - **write** slides
    - submit IRB request
    - **write** LSA abstract
    - run subjects
    - **write** grant proposal

Procedures in real life

- You don't specify in your To Do list what actions 'write' entails. To keep your To Do list manageable (and readable), you can just refer to the verb and leave the details to be specified somewhere else.

    - sit at computer
    - open text editor
    - be brilliant and eloquent
    - save (often)



Praat Procedures (manual)

- Praat Procedures are self-contained blocks of code that you may want to use multiple times

- They begin with: **`procedure`** `nameOfProcedure`

- They end with: **`endproc`**

- And you can then cause Praat to execute that named block of code anywhere in your script by typing: **`@nameOfProcedure: arg`**

Blue Monday

- In the day3Code/ folder there is a script called blueMonday.praat here is the important part of that code:

```
include split.proc.praat

song$ = "349.23, 261.63, 293.66, 293.66, 392.00, 261.63,
293.66, 293.66"

for reps from 1 to 2
     @split: ",", song$

     for note from 1 to split.length
          @playPairs: number( split.array$[note] )
     endfor
endfor
```

Blue Monday

- In the day3Code/ folder there is a scri
  important part of that code:

```
include split.proc.praat

song$ = "349.23, 261.63, 293.6
293.66, 293.66"

for reps from 1 to 2
     @split: ",", song$

     for note from 1 to split.length
          @playPairs: number( split.array$[note] )
     endfor
endfor
```

```
@procedureName( arguments )

We can call our new split
procedure to split up a comma-
delimited string.

.variables are available as
procedureName.variableName
```

Blue Monday

- In the day3Code/ folder there is a script called blueMonday.praat here is the important part of that code:

```
include split.proc.praat

song$ = "349.23, 261.63, 293.66, 293.66, 392.00, 261.63,
293.66, 293.66"

for reps from 1 to 2
     @split: ",", song$

     for note from 1 to split.length
          @playPairs: number( split.array$[note] )
     endfor
endfor
```

Split procedure by Jose J. Atria (full script)

```
procedure split (.sep$, .str$)
  # separate a string into tokens using a delimiter
  .length = 0
  repeat
    .strlen = length(.str$)
    .sep = index(.str$, .sep$)
    if .sep > 0
      .part$ = left$(.str$, .sep-1)
      .str$ = mid$(.str$, .sep+1, .strlen)
    else
      .part$ = .str$
    endif
    .length = .length+1
    .array$[.length] = .part$
  until .sep = 0
endproc
```

Split procedure by Jose J. Atria (full script)

```
procedure split (.sep$, .str$)
  # separate a string into tokens using a delimiter
  .length = 0
  repeat
    .strlen = length(.
    .sep = index(.str$
    if .sep > 0
      .part$ = left$(.
      .str$ = mid$(.st
    else
      .part$ = .str$
    endif
    .length = .length+1
    .array$[.length] = .part$
  until .sep = 0
endproc
```

```
procedure name (arguments)
      ... block of code
endproc
```

Split procedure by Jose J. Atria (full script)

```
procedure split (.sep$, .str$)
  # separate a string into tokens using a delimiter
  .length = 0
  repeat
    .strlen = length(.str$)
    .sep = index(.str$, .sep$)
    if .sep > 0
      .part$ = left$(.str$, .sep-1)
      .str$ = mid$(.str$, .sep+1, .strlen)
    else
      .part$ = .str$
    endif
    .length = .length+1
    .array$[.length] = .part$
  until .sep = 0
endproc
```

Split procedure by Jose J. Atria (full script)

```
procedure split (.sep$, .str$)
  # separate a string into toke
  .length = 0
  repeat
    .strlen = length(.str$)
    .sep = index(.str$, .sep$)
    if .sep > 0
      .part$ = left$(.str$, .se
      .str$ = mid$(.str$, .sep
    else
      .part$ = .str$
    endif
    .length = .length+1
    .array$[.length] = .part$
  until .sep = 0
endproc
```

Arrays (sort of)

If you have any experience programming in another language, you might expect Praat to have an array (or list) data type. It doesn't, really, but you can fake it.

Split procedure by Jose J. Atria (full script)

```
procedure split (.sep$, .str$)
  # separate a string into tokens using a delimiter
  .length = 0
  repeat
    .strlen = length(.str$)
    .sep = index(.str$, .sep$)
    if .sep > 0
      .part$ = left$(.str$, .sep-1)
      .str$ = mid$(.str$, .sep+1, .strlen)
    else
      .part$ = .str$
    endif
    .length = .length+1
    .array$[.length] = .part$
  until .sep = 0
endproc
```

Split procedure by Jose J. Atria (full script)

```
procedure split (.sep$, .str$)
  # separate a string into tokens using a delimiter
  .length = 0
  repeat
    .strlen = length(.str
    .sep = index(.str$, .
    if .sep > 0
      .part$ = left$(.str
      .str$ = mid$(.str$,
    else
      .part$ = .str$
    endif
    .length = .length+1
    .array$[.length] = .p
  until .sep = 0
endproc
```

Variable scope!

All variables in Praat are global –even in procedures. Starting a variable with (.) prevents the procedure from changing variables in surprising ways.

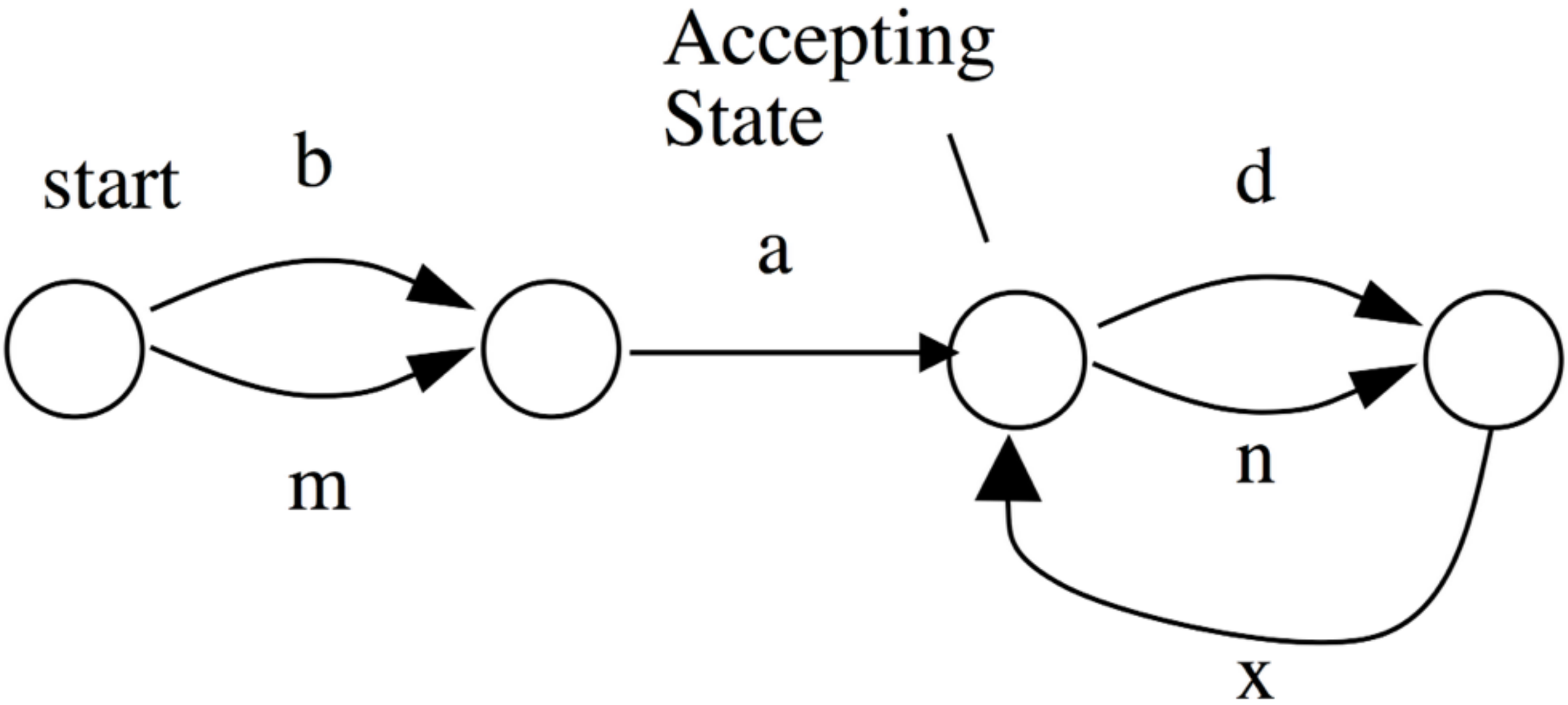
Split procedure by Jose J. Atria (full script)

```
procedure split (.sep$, .str$)
  # separate a string into tokens using a delimiter
  .length = 0
  repeat
    .strlen = length(.str$)
    .sep = index(.str$, .sep$)
    if .sep > 0
      .part$ = left$(.str$, .sep-1)
      .str$ = mid$(.str$, .sep+1, .strlen)
    else
      .part$ = .str$
    endif
    .length = .length+1
    .array$[.length] = .part$
  until .sep = 0
endproc
```

Exercise: Procedure

- An abbreviated version of a script Anita Szakay shared with me can be found in isVowel.praat
- Can you make the following changes to this code?

1. Add a form to prompt the user to type in some text
2. Use a procedure to test if the string is a vowel from Anita's list.
3. Can you rewrite the long test condition to use a string function instead?


Accepting
State
start
b
a
d
m
n
X

Regular Expressions

- Regular expressions are a powerful tool for searching
- A regular expression is a formal description of a family of strings
- Say you needed to find out whenever your Roman author did some loving, regardless of when it happened. You might want to match all and only the following strings:

  - amo
  - amabam
  - amabo
  - amavi
  - amaveram
  - amavero

Regular Expressions (manual)

- amo
- amabam
- amabo
- amavi
- amaveram
- amavero

am[oa]*

- This family of strings can be matched with the regular expression am[oa]*
- "am" will match any occurence of those two characters together.
- The square brackets define a **character class** that will match a single character.
- The * (Kleene star) will match zero or more characters up to the end of the line.

Regular Expressions

- Three string functions in Praat use regular expressions to find or replace strings: index_regex, rindex_regex, and replace_regex$

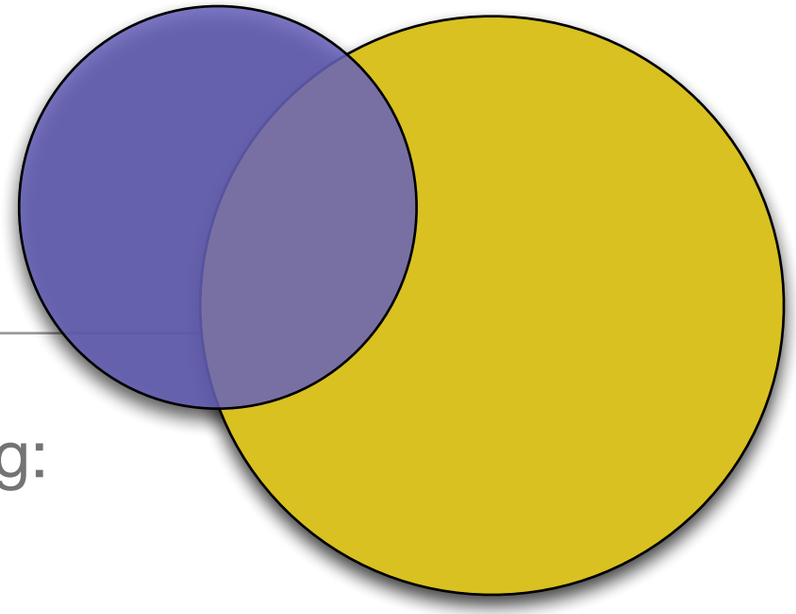
- `index_regex: string$, regex$` determines where the string `string$` first matches the regex `regex$. Declaring:`

```
match = index_regex: "internationalization", "a.*n"
```

- the variable `match` will contain the number 7. If there is no match, the result is 0.

- `rindex_regex: a$, b$` does the same from the right side of the string.

Exercise: regular expressions



- Please count the number of 's' characters in the string:

  she sells seashells by the seashore

- Your output should be the total count followed by a list of which positions in the string contain an 's'

- Advanced challenge: what if you wanted to count 's' and 'sh' separately?

Praat Scripting

Using other's code

Using others' code

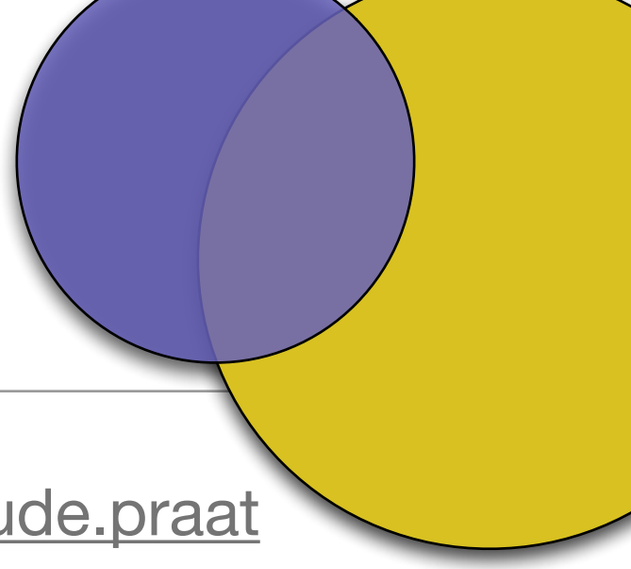
- One of the dirty secrets of software engineering is that most of a programmer's time is spent editing and maintaining existing code

- 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

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 day3Code/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 day3Code/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** - an unbelievably useful source of scripts, instructions, and step-by-step how to videos

- **Pauline Welby** - some useful scripts and an introduction

- **Kyuchul Yoon** - many, many useful scripts and an introduction in Korean

- **Holger Mitterer** - tons of good scripts, be sure to code review these first

- **Praat Script Archive** - a searchable index of many 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!