Praat Scripting
09 Procedures

Sometimes we want to use a portion of code in a script more than once. On the one hand, loops come in handy, on the other side this might not be enough for specific performances. This is where `procedures` come in handy.
A procedure is a block of several instructions which can be `called` whenever needed. Therefore, you can re-use similar pieces of code. Look at the following code...

```
key_x = 0.75
key_y = 490

selectObject:  "Sound " + name1$
To Pitch:  0, 75, 6000
selectObject:  "Pitch " + name1$
Black
Draw:  0, 0, 0, 500, "no"

selectObject:  "Sound " + name2$
To Pitch:  0, 75, 6000
selectObject:  "Pitch " + name2$
Black
Draw:  0, 0, 0, 500, "no"

selectObject:  "Sound " + name3$
To Pitch:  0, 75, 6000
selectObject:  "Pitch " + name3$
Black
Draw:  0, 0, 0, 500, "no"

drawing$ = "pitch_curves.pdf"
Save as PDF file:  drawing$
select all
Remove
```

As you can see, some arguments occur again and again. Using a procedure, you do not have to write those repeatedly but can re-use them whenever needed (see the following code).

```
key_x = 0.75
key_y = 490

soundname$ = "a"
Black
@draw

soundname$ = "b"
Red
@draw

soundname$ = "c"
Green
@draw

drawing$ = "abc.pdf"
Save as PDF file:  drawing$
select all
Remove

procedure draw
   selectObject:  "Sound " + soundname$
   To Pitch:  0, 75, 6000
   selectObject:  "Pitch " + soundname$
   Draw:  0, 0, 0, 500, "no"
endproc
```

As you see, a procedure definition in Praat consists of three parts:

1. a line with the word **procedure**, followed by the name of the procedure
2. the body of the procedure
3. a line with the word **endproc**

You can put a procedure definition anywhere in your script; the beginning or end of the script are common places. The bodies of procedures are executed only if you `call` the procedure explicitly (using the symbol `@` and the name of the procedure), which you can do anywhere in the rest of your script

Arguments

In the script above, you still have to define the single sound files that should be drawn and the color they should be drawn in. This can be improved. In the following version of the script, the procedure `draw` requires an explicit argument: `@draw:  "monotone", "Black"`. You can use multiple arguments, separated by commas, and string arguments (with a dollar sign in the variable name). For mere numeric arguments use something like `@draw:  400 + 100`.

```
key_x = 0.75
key_y = 490

@draw: "a", "Black"
@draw: "b", "Red"
@draw: "c", "Green"

drawing$ = "abc.pdf"
Save as PDF file: drawing$
select all
Remove

procedure draw: soundname$, color$
   selectObject: "Sound " + soundname$
   To Pitch: 0, 75, 6000
   selectObject: "Pitch " + soundname$
   color$
   Draw: 0, 0, 0, 500, "no"
endproc
```

This works as follows. The first line of the procedure now not only contains the name (draw), but also a list of variables (`soundname$` and `color$`). In the first line of the script, the procedure draw is called with the argument `"monotone"` and `"Black"`. Execution then jumps to the procedure, where the arguments are assigned to the variable `soundname$` and `color$`, which is then used in the body of the procedure.

Encapsulation and local variables

Look at the following script.

```
frequency = 300
@playOctave: 440
@playOctave: 400
@playOctave: 500
appendInfoLine: frequency
procedure playOctave: frequency
   Create Sound from formula: "note", 1, 0, 0.3, 44100, frequency, 0.2, 0.01, 0.01
   Play
   Remove
   octaveHigher = 2 * frequency
   Create Sound from formula: "note", 1, 0, 0.3, 44100, octaveHigher, 0.2, 0.01, 0.01
   Play
   Remove
endproc
```

You might have thought that this script will write "300" to the Info window, because that is what you expect if you look at the first five lines. However, the procedure will assign the values 440, 400, and 500 to the variable frequency, so that the script will actually write "500" to the Info window, because 500 is the last (fourth!) value that was assigned to the variable frequency.
What you would want is that variables that are used inside procedures, such as frequency and octaveHigher here, could somehow be made not to "clash" with variable names used outside the procedure. A trick that works would be to include the procedure name into the names of these variables:

```
frequency = 300
@playOctave: 440
@playOctave: 400
@playOctave: 500
appendInfoLine: frequency
procedure playOctave: playOctave.frequency
   Create Sound from formula: "note", 1, 0, 0.3, 44100, playOctave.frequency, 0.2, 0.01, 0.01
   Play
   Remove
   playOctave.octaveHigher = 2 * playOctave.frequency
   Create Sound from formula: "note", 1, 0, 0.3, 44100, playOctave.octaveHigher, 0.2, 0.01, 0.01
   Play
   Remove
endproc
```

Fortunately, Praat allows an abbreviated version of these long names: just leave "playOctave" off from the names of the variables, but keep the period (.):

```
...
procedure playOctave: .frequency
   Create Sound from formula: "note", 1, 0, 0.3, 44100, .frequency, 0.2, 0.01, 0.01
    ...
endproc
```