Music Creation with Python

Olivier Belenger

July 2020

Table of contents

Chapter 1

Installation guide

1.1 Installation of essential resources

First, since pyo is a Python module, a Python distribution must be installed on the system. Then, as pyo includes graphical elements written with WxPython, this library must also be installed under the Python distribution.

1.1.1 Install Python

Let's first install the official Python distribution, we will use version 3.7.4, say
available at this address:

https://www.python.org/downloads/release/python-374/

Windows

Under Windows, install the version that matches your computer's architecture (most likely 64-bit), x86 for a 32-bit processor or x86-64 for a 64-bit processor. This
are the files:

32-bit: Windows x86 executable installer.

64-bit: Windows x86-64 executable installer.

Warning: When installing, choose the Customize Installation option, then leave the configuration as it is but make sure to check the Add Python to environment variables box.

macOS

Under MacOS, install the version for 10.9 by mounting, it is the file identified as follows:

Mac OS X 64-bit install.

Follow the installation instructions!

1.1.2 Install WxPython

As pyo includes some graphical interface elements written with WxPython, it is now necessary to install this library (version 4.0.7, pyo 1.0.1 is not compatible with version 4.1.0). Installation is done with Python's built-in package manager: pip.

Windows

Open a command prompt (Command Prompt, cmd in search) and enter the following line, followed by the Return key:

```
pip install -U wxPython==4.0.7
```

macOS

Open the Terminal application and enter the following line, followed by the Return key:

```
sudo pip3 install -U wxPython==4.0.7
```

1.1.3 Install pyo

Pyo is also hosted on pypi.org and installed with the official python package manager, pip:

Windows

Open a command prompt and enter the following line, followed by the Return key:

```
pip install -U pyo
```

A specialized text editor, EPyo, for audio scripting is installed along with pyo. If you checked the Add Python to environment variables box when installing python, you can launch the editor by running the following command in a command prompt:

```
epyo
```

The Windows platform is particularly capricious when it comes to audio drivers.
There are several and not all of them work! If you have problems with pyo's audio output, it's a matter of finding the driver that works well on your system. Refer to the Windows Audio System Inspection man page for guidance:

http://ajaxsoundstudio.com/pyodoc/winaudioinspect.html

macOS

Under MacOS, pyo is also installed via the official python package manager, pip. In a terminal window, run the following command (you will have to provide your password):

```
sudo pip3 install -U pyo
```

A specialized text editor, EPyo, for audio scripting is installed along with pyo. You can launch the editor by running the following command in a terminal window:

```
epyo
```

linux

Under major linux distributions, pyo can also be installed via the package manager python official, pip. In a terminal window, run the following command:

```
sudo pip3 install -U pyo
```

1.1.4 Test the working environment

You are now ready to run a first pyo script. Open the EPyo application, select an example from the Pyo Examples menu (the file will show up in the text editor) and activate the Run command by pressing Ctrl+R (Cmd+R on MacOS).
The Audio Server window should appear. Press the Start button to activate the audio.

1.1.5 Online Documentation

The pyo online manual can be consulted at the following address:

http://ajaxsoundstudio.com/pyodoc/

Chapter 2

Introduction to Python programming language

2.1 General concepts of programming

A computer does nothing but perform simple operations on a sequence of electrical signals. These signals can take only two states, a minimum or a maximum electrical potential. We will generally consider these signals as a sequence of numbers having the value 0 or 1. A numerical system thus limited to two digits is called a binary system.

Any information of another type will therefore have to be converted into binary format to be processed by the computer. This is true not only for all types of files, such as texts, images, sounds or films, but also for the different programs, i.e. the sequences of instructions, that 'we want the computer to execute in order to process data.

These binary numbers form a long sequence of 0s and 1s and are generally processed in groups of 8 (byte), 16, 32 or 64, depending on the system used. Automatic translators are therefore necessary in order to convert the instructions made up of strings of characters, forming keywords, into a series of binary numbers.

The translator will be called an interpreter or a compiler, depending on the method used to perform the conversion. The set of keywords, associated with a set of well-defined rules, will constitute the programming language. The rules will indicate to the translator how to assemble the directives provided by the keywords in order to compose instructions understandable by the machine, ie a sequence of binary numbers.

Depending on the type of instructions it contains, a language will be said to be low level (eg Assembler) or high level (eg Pascal, Perl, Java, Python, etc.). A so-called low-level language is made up of very elementary instructions, very close to machine language. The high-level language will offer more complete and powerful commands. A single command in a high-level language will result in several elementary instructions. It will be the role of the interpreter or compiler to translate these instructions into machine language. High-level languages are always much easier to learn and use since they do a lot of the programming work.

Different types of programming:

— Sequential programming: All lines of source code are evaluated one after the other, from the first to the last. To execute a piece of code again or to skip several lines, it is necessary to resort to instructions such as GOTO or JUMP. This type of programming is very difficult to read since it goes in all directions. Csound is somewhat like sequential programming.

• Procedural programming: Is based on the procedure call paradigm. A procedure, also called a function, method or routine, consists of a sequence of commands to be performed. It is possible to call any procedure at any stage of the program. One of the advantages of procedural programming is the possibility of reusing pieces of code several times without having to rewrite them each time.
It also allows the creation of modular and structured programs, easier to read than sequential programming. The languages Perl, Basic and C are examples of procedural programming.

— Object-oriented programming: The basis of this type of programming is the object. An object is a data structure responding to a set of messages. The data that describes the structure of the object is called its attributes, while the response to receiving a message is called a method. Objects are defined by classes, which can inherit attributes and methods from other classes. This type of programming, more modern, is very powerful in that it allows the compartmentalization and the redefinition of all the elements constituting the program. Java, C++ and Python are object-oriented programming languages.

2.2 Introduction to Python

Python is a young programming language, dynamic, free and supported by a large community of programmers. It is highly extensible, ie adding libraries, written in Python, C or C++, is extremely simple. It is an interpreted language, supporting the modular and object-oriented approach to programming.

Main characteristics of the language:

— Python is cross-platform, fully supported on major operating systems
tation (Mac OS, Windows, MS-DOS, Unix, etc).

— Python is free and open source , it is therefore permitted to use and modify the code
software.

— Python is efficient for writing small programs of a few lines as well as
very complex programs of several tens of thousands of lines.

— Python offers an extremely simplified syntax, as well as highly advanced data types (lists, dictionaries and others), making it possible to write very compact programs,

readable and effective. A Python program is generally 3 to 5 times shorter than the corresponding program written in C.

— Python automatically manages its resources (memories, file descriptors). It is therefore not necessary to keep references and to ensure that unused objects are indeed destroyed.

— Pointers do not exist in Python, which greatly simplifies the writing of pro grams.

— Python is an object-oriented language supporting multiple inheritance and polymorphism, ie the reuse of the same code with different data types.

— Python's exception handling system is simple to use and very efficient.

— Python is dynamically typed. Every manipulable object has a well-defined type that it is not necessary to have previously declared.

— Python is an interpreted language. Programs are compiled into portable instructions and then executed by a virtual machine.

* The last two features necessarily imply extra operations for the processor to run the program correctly. This results in a slower execution speed compared to a compiled language.

2.3 Compilation and interpretation

The program written by the programmer constitutes the source code of the application. To run this program, the source code must be translated into binary code executable by the machine. As mentioned previously, there are two techniques for performing this translation: interpretation and compilation.

When a program is interpreted, no object code is produced. The interpreter parses each line of the program and transforms it into machine language instructions which are immediately executed.

Source code =ÿ Interpreter =ÿ Result

Compiling a program consists in translating all the source code in advance and creating an object code, whose language is much closer to that of the machine. This becomes an executable file at any time, without the need to go through the compiler again.

Source code =ÿ Compiler =ÿ Object code =ÿ Executor =ÿ Result

Interpretation is ideal in a learning or program development context, as it allows immediate testing of changes to the program without having to recompile the code. source. On the other hand, as the language must be translated in real time by the interpreter, certain operations can take a long time and

not be executed fast enough for the programmer's liking. In this case, a compiled program will always be the fastest and most efficient option, since the object code speaks a language much closer to machine language.

Some modern languages, such as Python, attempt to bridge the two techniques.
When Python receives a source program, it first compiles it to produce intermediate code (bytecode), closer to machine language, which will be passed to the interpreter for execution. Interpretation of bytecode is faster than that of source code.

Source code =ÿ Compiler =ÿ Bytecode =ÿ Interpreter =ÿ Result

Advantages :

— The bytecode is portable, it suffices to have a Python interpreter on the machine to run any code.

— Interpretation of bytecode is sufficiently fast, although slower than real object code, for the vast majority of applications.

— The interpreter is always available in order to quickly test small bits of pro gram.

2.4 The different types of errors

Executing a program is an extremely capricious operation. The slightest mistake can cause the execution to stop. Three types of errors can be encountered when designing a program:

— Syntax errors: All programming languages have their own syntax.
This is generally very precise and it is important to respect the rules when writing programs. The slightest syntax error, such as a misplaced comma, will cause the program to stop and an error message to be displayed. It is very important to understand the structure of error messages, because these usually give us very useful clues to correct the error that has been made. For example, Python will tell us if the error is a syntax error and at what line of the program it was detected.

— Semantic errors: Semantic error, or logic error, occurs when the specified instructions are valid but do not correspond to the original idea of the programmer. The program therefore runs perfectly, but the result obtained is not the one expected. These errors are more complicated to correct. You have to do a detailed analysis of all the parts of the program to know which piece of code does not give the expected result. There are several Python tools to help with debugging, we'll cover a few in time.

— Runtime Errors: These errors mainly occur when a program,
functioning normally, tries to execute a command which, for some reason, is no longer possible.
For example, the program is trying to read a file that has been
deleted from the hard disk, or it tries to import a library that is not installed on the
system. These errors are called exceptions, and cause the program to stop.
We will see various ways to handle exceptions, in order to allow our program
to continue driving even when this situation occurs.

2.5 Variable names and reserved words

The work of a computer program consists mainly of manipulating data
in order to achieve a certain result. This data is stored in the language using
of variables. These variables can be of different types (which we will see later) and
have an arbitrary name, given by the programmer. These names must be the most evocative
possible, in order to facilitate the reading of the program. Although we access a variable by
calling it by its name, it is in fact only a reference designating a memory address,
that is, a location in the computer's RAM. This location contains the
real value given to the variable in binary format. Any object can be placed in
memory, be it an integer, a real number, a character string, a list, a
dictionary or whatever. In order to distinguish these different contents, the programming language
uses variable types, real integer etc. This topic,will be covered in listing ,
the next chapter.

The names given to variables are left to the programmer. They must, however,
respect a few rules, here are those in force in the Python language:

1. A variable name can be made up of letters and numbers, but must always be
start with a letter.

2. The only special character allowed is the underscore (_). Accented letters, c´edilles,
spaces and special characters such as #, $, @, etc, are not allowed. The reason is that
these characters are already part of the Python syntax and have a special role to play.
For example, the character # indicates the beginning of a comment, i.e. a line that
will not be executed by the program.

3. Case (upper and lower case) is significant. Tempo and tempo are therefore two vas
distinct laughs!

By convention, variable names begin with a lowercase letter and do not use
capital letters only to increase the readability of the code (ex: listOfScales). It is very important
that a variable name be as evocative as possible, in order to make your code more readable for
others.

In the Python language (version 3), there are 33 reserved words that cannot be used
as a variable name since they are already used by the language itself and each have a
own functionality. They are listed in the table on the next page.

and ace assert break class else if keep on going
def del elif from global except finally
for is lambda nonlocal not try import in
gold pass
raise return while with yield
NoneTrue False

2.6 Practice

2.6.1 First steps in writing an audio script

Any script wishing to use the functionality of the pyo library must first import the module. We will come back later on the different ways to import a module, for the moment we will use the following line to have direct access to all the classes and functions present in the library. The star signifies all the elements!

```
from pyo import ÿ
```

Another essential step: Create an audio server. The server ensures the audio communication with the sound card and it is also the server that manages the clock, that is to say the scrolling of the samples in time. An audio engine always works by group of samples, ie it calculates a whole block of samples for a given object before calculating the block of the following object. The length of this block is called buffer size. The buffer size will affect the CPU load as well as the system latency, ie the delay induced by the calculation of the sample blocks. At first, we will work with the server defaults, which we will initialize as follows:

```
s = Server ( ) . boot()
```

The Server object with no parameters will use the values defined by default. The boot method (we will detail the classes and methods later) tells the server to initialize the audio communication. The server clock is not yet started but it is now ready to host objects performing sound processes. It is essential that a server exists before creating pyo objects!

`

From the moment a server is activated, all the pyo objects that will be created will be automatically counted by the server, which will calculate the sample blocks according to the order of creation Objects. This is where we can define our signal processing chain.
Let's start with the classic example, a sine wave at 1000 Hertz:

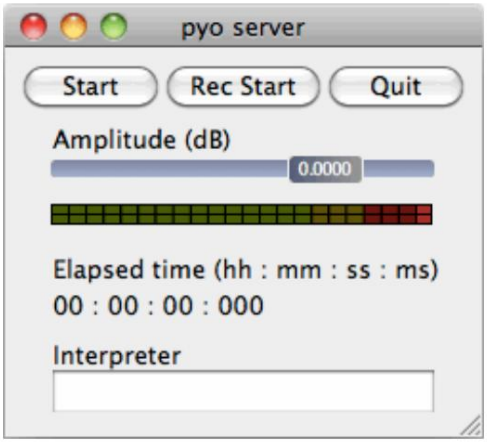
```
a = If not ( ). out ( )
```

We'll get to the basics of audio objects in a moment! We now have a server ready to calculate a signal, we just have to start its internal clock.
We will use the graphical interface included in the Server object to facilitate this task. We call this interface with the gui method:

```
s. gui( locals ())
```

`

When running the script, this line will cause the window to appear offering basic server controls, such as starting and stopping the clock, a save-to-disk button drive and an overall volume control. We will see in time the usefulness of adding locals() inside the parentheses.



Our first script will therefore look like this:

```
1 from pyo import ÿ 2 s =
S erver ( ) . boot ( ) 3 a = If
not ( ) . out() 4s . gui( locals
())
```

scripts/chapter 03/01 play sine.py

2.6.2 The PyoObject object

The PyoObject object is the base (parent class) of almost all the objects available in the library. The functionalities of this object can therefore be applied to all the objects that result from it (child class). Later, we'll explore class inheritance in more detail. For now, consider that any object, if its man page contains the following line,

Parent: PyoObject

has access to the functionality of this class.

Methods

Some methods that apply to the majority of objects in the library:

- obj.play(): The play method starts calculating samples of the given object. To note that play does not send sound to the audio output.
- obj.stop(): The stop method stops the calculation of the samples of the given object. She permits to save CPU if the object is not used for a certain period of time.

obj.out(): The out method starts the calculation of the samples of the given object and also indicates that the signal generated by the object should be routed to to the audio output. If an integer is given between the method parentheses, it indicates the output channel, starting at 0. In a stereo system, 0 indicates the output channel. left and 1 indicates the right channel.

- obj.ctrl(): The ctrl method displays a window where potentiometers are assigned to the control parameters of the given object. This window will allow us, first of all, to quickly explore the impact of the parameters on the generated sound signal.

In the previous script, insert the line

```
at . ctrl ( )
```

just after initializing object a, will display the following window:

2.6.3 How to read the pyo manual

The manual pages are separated by subject. Each page can contain up to 6 sections. Here is a brief description of each section:

Section Title

The first line following the name of the object is always the initialization line, i.e. the moment when an object is created and assigned to a variable . It contains the parameters governing the behavior of the object as well as their default value, if any. Next comes a description of the process performed by the object, then its parent class, if any. If the object has a parent class, all the functionalities of this class apply to the current object.

Parameters section

Detailed description of each of the object's parameters. Depending on the name of the parameter, there are always the possible types for the latter (ex: PyoObject, float, string, etc.). It is very important to respect the allowed types for a parameter. If there is no mention optional, it means that it is mandatory to provide a value to the parameter. The input parameter of objects that transform the signal (effects) is a good example, the object expects a signal to transform!

Mandatory parameter whose only type allowed is the PyoObject:

```
input: PyoOb ject Inpu
       tsignal .
```

Optional parameter accepting real numbers (float) or PyoObjects. If the parameter is not specified, the value -7.0 will be used:

```
transpo: float PyoObject, op ti on al T ranspo si ti
        onfactorin semi t one. D efa ul tsto ÿ7.0.
```

Notes section

If the object has some peculiarities of its own, they will be detailed in this section. For example, it is here that will be specified if an object does not have an out method (the signals that we do not want to send to the loudspeakers) or the attributes mul and add.

Example section

This section provides an example of using the object in a simple process. The script can be called from the command line as follows:

```
>>> from pyo import ÿ >>>
example ( Harmonizer )
```

Do not hesitate to take inspiration from it!

Methods section

Here are listed all the methods allowed on an object (to which you must of course add the methods of the parent class!).

```
a = Harmonizer ( input ). out() a. se tT
r an sp o ( 5 )
```

a.setTranspo(5) replaces the object's current transpose value and sets it to 5.0.

Attributes Section

Gives the list of attributes of an object (in addition to the attributes of the parent class). An attribute can be called directly on an object without going through the associated method. We will return later to the use of methods and attributes.

The following example performs the same operation as the previous example.

```
a = Harmonizer ( input ). out() a.
transpose = 5
```

Chapter 3

Operations, variable types and instruction flow

3.1 Compute with Python

In the Python language, a number can be expressed in different formats. It can be either an integer, with 32 bit resolution (or more if needed) or a real number, often called a floating point number.

— Integer: 12, -234, 0, ...

— Float: 0.005, 13.4654, -4.321, ...

Keep in mind that the precision of a number is dependent on the amount of bits used to represent that number. In the case of real numbers, the value used will almost always be an approximation of the requested value. If you type the number 3.23456 into a Python interpreter, this is what you might get (the number of decimal places displayed depends on the system):

```
>>> 3.23456
3.2345600000000001
```

Python is still an extremely precise computing environment, real numbers are always represented with 64 bits. We note that the approximation is really very close to the requested value. Python uses standard operators to express mathematical formulas:

— Multiplication: — *

Division (decimal numbers): / (5/2 = 2.5)

— Division (integer): / (5//2 = 2 or int(5/2) = 2)

— Addition: +

- substraction : -

In Python 3, a division always returns a floating point number. The symbol // can be used to obtain a division which returns an integer. This symbol does not exist in Python 2, however, the int(5/2) syntax is compatible between the two versions.

Two other operators are available, the exponent and the modulo.

— Exhibitor: ** (2**4 = 16)
— Modulo: %

The modulo operator returns the remainder of a division of one number by another.

```
>>> 3% 2 1
 >>> 12% 5


2 >>> 12% 6
0
```

In the Python environment, the standard order of precedence is respected. It can be memorized with the mnemonic expression PEMDAS (parenthesis, exponent, multiplication, division, addition, subtraction). Within the framework of a complex expression, the brackets are always calculated first, then it is the exponents, then the multiplications and the divisions, which have the same order of priority, and finally the additions and the subtractions. A simple careless mistake in the order of priorities can lead to very different results:

```
>>> 3. 1 4 1 5 9 ÿ 4 ÿÿ 2 . 5 0. 2 6 5
 4 4
>>> ( 3. 1 4 1 5 9 ÿ 4 ) ÿÿ 2 . 1 5
 7. 9 1 3 4 0 6 4 9 6
```

3.2 Typing of variables

In several languages such as C or Java, it is mandatory to declare the variables that one intends to use, as well as their type (that is to say that the typing is static), before assigning them a content. A variable cannot change type during execution and will not accept a value that does not match its type. Here is an example in C where we have two variables that we want to add together:

```
int a = 5; int
b = 7; int
result;

result = a + b;
```

Under Python, the typing of variables is dynamic, i.e. Python loads automatically assign the type that best matches all specified variables in your program. The same operation in Python would give:

```
a = 5
b = 7
result = a + b
```

The integer type will be automatically assigned to the variables a and b while the variable result will be assigned a type according to the result of the addition (in this example, an integer). This method is a bit slower to execute, since whenever Python encounters a variable, it must determine its type. On the other hand, it is extremely flexible and will allow manage highly sophisticated data structures such as lists and dictionaries.
We will come back to these types of data later. Thus, under Python, a variable can very well contain an integer at the beginning of the program and, along the way, become a list or a string.

3.3 Main types of data

In Python, you can always ask the interpreter about the type of a variable given. The main data types available in the language are:

— None: Type of the null value None

— int: An integer

— float: A real number

— str: A character string in text format

— bytes: A string of characters in binary format

— tuple: A list in brackets (1, 2, 3, 4)

— list: A list [1, 2, 3, 4]

— dict: A dictionary

The type function lets you know what the type of a variable is:

```
>>> type ( 1 )
<classint'>
>>> type(1.1)
<class 'float'>
>>> type ( ," hello " )
<classstr '>
>>> type ( b" hello " )
<class 'bytes'>
>>> type ( ( 1 2 ),)
<classtu pl e '>
>>> type ( [ 1 2 , , 3 , 4 ] )
```

```
<class 'list'>
>>> type({ ”shift”: <class [0, 4, 7]})
'dict'>
>>> type(1.1) isfloat
True
>>> type ( ” hello ”) isstr
True
>>> type (b”hello”) isbytes
True
```

The string type in Python has a particular syntax. It can be delimited by single quotes (') or double quotes (”). Both syntaxes are quite equivalent but it makes it easy to introduce a string into a string!

```
>>> ' This is a thong ',
' This is a thong ”
>>> ” This is also a thong ',
' This is also a thong „ '
>>> ' This is a string ” within a stringstring ” within a string
„ ' This is a
```

3.4 Assignment of values

The assignment of a value to a variable is done with the symbol '='. It is in no way case of an equality in the mathematical sense of the term. It just means that we have dropped the content x into the variable y. The variable name represents a memory space in the computer where the assigned value is stored in binary format. To test the relationship equality between two variables, we use the symbol '==', which is part of a group of symbols called comparison operators, we will come back to this.

```
n = 10
```

When we assign a value to a variable, as in the example above, this instruction has the effect of carrying out several operations in the memory of the computer:

1. Create and store a variable name
2. Assign it a specific type
3. Create and store a particular value at the assigned location
4. Establish a link (by an internal system of pointers) between the name of the variable and the memory space of the value

Multiple assignments

In Python, you can assign the same value to several variables at the same time:

```
a=b=c=1
```

Three variables, a, b and c, have been created and all have the value 1.

Parallel assignments

You can also assign several values to several variables at the same time:

```
at , b = 0 , 1
```

The value 0 has been assigned to variable a while the value 1 has been assigned to variable b.

3.5 Sequence of instructions and conditional execution

3.5.1 Sequence of instructions

A computer program is always a series of actions to be performed in a certain order. The structure of these actions and the order of execution constitute an algorithm. The structures of control are instructions that determine the order in which actions will be performed. He There are three types of structures: sequence, selection and repetition. We will develop on rehearsal in the next lesson.

If your program does not contain select or repeat statements, the lines will be executed in order, from the first to the last. When Python encounters a conditional statement, for example the if statement, it may take different paths depending on the result of the condition.

3.5.2 Selection or conditional execution

The execution of a complex program will be influenced along the way by various factors, such as the answers provided by the user to certain questions or the state of certain program variables. It is with conditional statements, placed at specific places, that the program is specified to take this or that path depending on the state of the data. A conditional statement begins a line with the keyword if and ends it with the symbol two points (:). This symbol marks the beginning of a control sequence which can be a condition, a loop, the definition of a function or a class, etc. The next line must be indented to specify to Python that this statement should only be executed in certain circumstances.

By convention, the indentation corresponds to 4 empty spaces left at the beginning of the line. Done attention, the tabulation key, depending on the editor you are using, generates either a character TAB (usually equivalent to 4 blank spaces), i.e. 4 space characters. These two character types are not encoded in the same way by the language and can create conflicts since Python will not consider lines at the same level of indentation, even if Visually everything looks flawless. Most smart editors have the option to automatically replace the TAB characters by 4 empty spaces.

Here is a simple if statement, note the prompt change from the Python interpreter at the beginning of the line!

```
>>> a = 1
>>> ifa == 1:
 . . .          print ( ' a is 1            ' )
 . . .
a is 1
```

The three dots, which we call secondary prompt, mean that we are entering a block statements that we need to indent. Since the condition has been true, the next line that of the if has been executed and the interpreter has displayed the value 1. If the condition had turned out to be false, that line would simply not have been executed.

It is possible to specify another path, with the else statement, in case a condition is false:

```
>>> a = 0
>>> ifa == 1:
 . . .          print('a is 1 . . . else:    ' )

 . . .          print('a is not 1')
 . . .
a is not 1
```

Note the colon after the if and else statements to signify entry into a block of instructions to indent. Our program now defines two possible paths by depending on the state of variable a at the time of executing these lines. The elif statement allows to make a condition more complex and to define several different actions according to several conditions:

```
>>> a = 0
>>> ifa > 0:
 . . .          print('aestpositive')
 . . . elifa < 0:
 . . .          print('aisnegative')
 . . . else:
 . . .          print ( ' is neither positive nor negative          ,  a is therefore 0 ' )
 . . .
is neither positive nor negative          ,  a is therefore 0
```

A condition can contain as many elif as necessary, but only one if statement and a single else statement.

3.6 Comparison operators

Comparison operators are used to test the relationship between the different variables of a program. Depending on the result obtained, we can decide to bifurcate the flow of instructions to operate in one way rather than another. The different comparison operators are :

== equality

! = inequality

< smaller than

> greater than

<= less than or equal

>= greater or equal

Examples:

```
a = 10
if a <= 1 0:
        print ( " )  a is smaller or eg al a 10 elif a > 1 0:

        print ( " )  a is greater than 10 else :

        print ( 'If one day you get this rep we                    ,   let me know! ' )
```

Later in the session, with the knowledge you will have acquired, come back to this script and try to get the answer following the else statement!

```
a = 7
if ( a % 2 ) == 0:
        print ( 'ais an even number' )
        print ( 'because the remainder of its division by 2 is 0' )
else :
        print ( 'ais an odd number' )
```

Note the sign (==) to test for equality between two values. The equal sign alone (=) is used for assigning a value to a variable.

3.7 Compound Statements

The last example on the comparison operators constitutes a first block of instructions, or compound instructions. Under Python, there are two essential rules to respect for build a block of instructions:

1. The line that announces the block, the header, always ends with a colon.
   The following keywords announce a block of instructions: if, elif, else, for, while, def, class, with, try, except, finally.
2. The lines that make up the block of instructions must be indented at exactly the same level (count 4 empty spaces).

```
a = 5
if a > 0: # lined ' header
        print ( 'aestpositive' )
        print ( " )  because it is greater than 0 else :

        print ( 'isisnegative' )
        print ( 'be null' )
        print ( " )  because it can be smaller or eg al to 0
```

All lines of a block of instructions must absolutely be at the same level of indentation. These instructions are called a logic block, because each of them will always be executed only if the condition allows it. Whatever happens, they will always suffer the same fate.

A quick reminder: don't forget that Python does not encode a character in the same way. tabulation and a space character !

3.8 Nested statements

In order to achieve complex decision structures, you will probably have to nest into each other several compound statements.

```
if a >= 0:
        if ( a % 2 ) == 0:
                if a < 1 0 0:
                        print ( ' a is a value eurp ai enters 0 and 100 ' )
                else :
                        print ( ' a is a value greater than 100 ' )
                        a = a % 100
        else :
                if a < 1 0 0:
                        print ( 'a is an odd value, between 0 and 100' )
                        a = a + 1
                else :
                        print ( ' a is an odd value greater than 100 ' )
                        a = ( a % 1 0 0 ) + 1
else :
        print ( 'aisnegative' )
        a = 0
```

- What are the possible values for the variable a at the output of this block of instructions nested?

3.9 Syntax rules

In Python, the layout of your program defines block boundaries instructions. This rule, in addition to greatly simplifying the syntax, obliges the programmer to write clean code that is easy to read for someone who is not the author. Some languages such as C or Java use symbols to delimit blocks, usually braces. This avoids worrying about the indentation of the program and often generates code that is virtually unreadable except for the programmer who wrote it. In Python, these are the levels of indentation which delimit the blocks constituting the conditions, the loops, the functions and other compound statements.

```
# Block 1
 . . . .
lined'          header:
        # Block 2
         . . . .
         header line:
                # Block 3
                 . . . .
        # Block 2 (continued)
         . . . .
# Block 1 (continued)
 . . . .
```

In Python, an instruction line ends with the carriage return (which avoids the use of the semicolon, encountered in many languages!). We will see later how to extend a statement over several lines. It is also possible to end a line with a comment. A comment is anything between the pound sign (#) and the carriage return. This text serves as a guide or explanation to the different parts of the program and will be completely ignored by the interpreter at runtime.

It is very important to comment the program well at all stages of the construction, in order to find your way back to it a few years later when you have reorganized your code.
Comments are very valuable guides for a programmer who, having not participated in the development, would like to add his touch to the program. He must be able to quickly understand which directions the flow of instructions is taking.

Note: A good comment is not one that says how it's done... but rather one that says why it's done! We will have the opportunity to discuss it again.

3.10 Exercise

Calculation and condition

Knowing that the temperature conversion is done as follows:

C to F ÿ > we multiply by 9, then we divide by 5, then we add 32
F to C ÿ > we subtract 32, then we multiply by 5, then we divide by 9

1. Assign a value in Fahrenheit to a variable f.
2. Then perform a conversion from Fahrenheit to Celsius, assign the result to a variable c and display the temperature in Celsius.
3. Then, build a block of conditional statements that will output one of the following sentences, depending on the temperature in Celcius:

   - Probably ice cream.

   — In a boiling state.

   — This “water” is neither too hot nor too cold.

Attention, the mathematical precision is of rigor as well as the respect of the syntax of the sentences to display. In addition, your script must work flawlessly in all cases. Test it well!

Try to complete the exercise before consulting the solution on page 37.

3.11 Practice

3.11.1 Create a signal processing chain

All objects used to modify the signal (eg: filters and effects) have a first parameter called input. An object of type PyoObject must absolutely be given to this parameter.
The modifier object will take the signal from the input object and apply a transformation algorithm to it to generate its own audio signal. This signal can be sent to the audio output, or to another transformation object. Example :

```
1 from pyo import ÿ 2 s = S
erver ( ) . boot() 3 fm = FM( ) 4
fm . ctrl() filt = ButLP( fm).
out(0) 6 filt. ctrl() 7 harmo =
5   Harmonizer (filt). out(1) 8 harm.
ctrl() 9 sec. gui( locals ())
```

scripts/chapter 03/02 dsp chain.py

Objects to explore:

— Generators: Sine, FM, SineLoop, LFO, Rossler, SuperSaw, SumOsc, RCOsc

— Filters: ButLP, ButHP, ButBP, ButBR, Biquad, MoogLP, Reson, SVF

— Effects: Chorus, Disto, Freeverb, WGVerb, Harmonizer, FreqShift

— Pan: Pan

Create various processing chains, consisting of a generator followed by one or more effects, and explore with the knobs to familiarize yourself with the system and the sounds.
Pay attention to volumes!

3.11.2 Conditional statements

It is common to use a conditional statement in order to steer a program in a particular direction by modifying only one variable. For example, instead of modifying the program for each new source that we want to explore, we can create a conditional instruction where each of the sources will correspond to a particular condition. The conditions will be performed on integers:

```
src = 1

if src == 0:
        source = If not ( ). out ( ) elif
src == 1 : source = FM( ) . out ( )
```

```
elif src == 2:
        source = SineLoop (). out ( )

source . ctrl ( )
```

Depending on the value given to the src variable before executing the script, the source variable will represent a sine wave, frequency modulation or looped sine wave.
If we apply the same principle to effects, we get:

```
1 from pyo import ÿ
2s = Server ( ) . boot()
3
4 src = 1
5 fx = 0
6
7   if src == 0:
8         source = If not ( ). out ( )
9   elif src == 1:
10        source = FM( ) . out ( )
11  elif src == 2:
12        source = SineLoop (). out ( )
13
14 source . ctrl ( )
15
16  if fx == 0:
17        effect = Chorus (source). out ( 1 )
18  elif fx == 1:
19        effect = Harmonizer (source). out ( 1 )
20  elif fx == 2:
21        effect = Di sto (source). out ( 1 )
22
23 effect . ctrl ( )
24
25 sec. gui( locals ())
```

scripts/chapter 03/03 conditions.py

3.11.3 Man pages to consult

server	PyoObject	sine	FM
SineLoop	LFOs	SuperSaw	SumOsc
RCOsc	ButLP	ButHP	ButBP
GoalBR	biquad	MoogLP	Reson
SVF	WGVerb	Chorus	Distortion
Freeverb	harmonizer	FreqShift	Pan

3.11.4 Examples and exercises

All the exercises proposed in these pages are intended to guide you in the development a personal library of sound processes that you will use when making the works. It is therefore very important to follow the steps conscientiously, in order to master

the different techniques, and to keep all your attempts. You will have to regularly revisit these bits of code to flesh them out.

The one and only effective recipe for learning a programming language is practice. Taking the time to understand and assimilate these simple exercises will make performing more complex (and therefore more interesting!) processes seem much easier.

Exploration of sounds from elementary sources of synthesis.

Exercises:

By using the objects proposed above (generators, filters and effects from the Pre first step), explore various processing chains of your composition.

1. You can create as many objects as necessary, i.e. multiple sources and/or several effects connected in cascade or in parallel.
2. Use the default values for object initialization. The only argument to specify is the audio input for the effects. You can also specify an output channel to the out() method. Example, out(1) to route the signal to the right speaker.
3. Display the control window (with the ctrl() method) to vary the value of the different parameters.
4. Pay attention to the volume! The default behavior of objects is to play at full power. For now, we'll just reduce the server volume, which affects the final signal. Observe how the following example initializes the server to an amplitude of -20 dB!

5. You are free to keep all your experiments in one script and navigate between them using a condition (as in the following example) or work on files independent, it is according to your ease.

Examples:

The following script proposes three processing chains with different connection schemes. The rst example consists of a source and two effects connected in a cascade where only the final result is heard. The second proposes a source connected in parallel to two effects (each of the effects is routed to a loudspeaker) and the last is built with three sources added at the input of a single effect.

```
"""
Examples of processing chains. No initialization parameter.
Ex pl ora ti on with the window

Example no 1: C onnec ti on s in cascade:
        sine ÿ> distortion ÿ> chorus

Example no 2: C onnec ti on s in parallel (multiÿeffects)
        sineloop ÿ> freqshift
                    ÿ> freqshift

Example no 3: C onnec ti on s in parallel (multiÿsources) lfo 1 ÿ>

        lfo 2 ÿ> ha rm onize r
        lfo 3 ÿ>
"""

from pyo import ÿ

s = S erver () . boot()
sec. amp= . 1 # initialize the server to ÿ20 dB

example = 2 # modify this variable to change example

if example == 1:
        src = If not ( )
        src. ctrl ( )
        dist = Di sto ( src )
        dist. ctrl ( )
        ch or = Chorus (dist). out ( )
        ch or . ctrl ( )
elif example == 2:
        src= SineLoop ()
        src. ctrl ( )
        fshif = Freq S hi ft ( src ) . out ( )
        fshif. ctrl ( )
        fshif 2 = Freq S hi ft ( src ) . out ( 1 )
        fshif 2 . ctrl ( )
elif example == 3:
        src 1 = LFO( ) . out ( )
        src 1 . ctrl ( )
        src 2 = LFO( ) . out ( )
        src 2 . ctrl ( )
        src 3 = LFO( ) . out ( )
        src 3 . ctrl ( )
        harm = Harmonizer ( src 1+src 2+src 3 ) . out ( 1 )
        harm. ctrl ( )

sec. gui( locals ())
```

scripts/chapter 03/04 examples.py

3.11.5 Solution to the temperature conversion exercise

```
f=212

c = (f ÿ 3 2 ) ÿ 5 . / 9

prints ( c )

if c <= 0:
        print ( 'Surely ice cream' )
elif c >= 1 0 0:
        print ( ”Boiling else :                    ” )

        print ( ' This t te  “water” is neither too hot nor too cold scripts/chapter      ' )
```

03/05 exercise.py – –

Chapter 4

Data structures

4.1 Character strings: functions specific to the string type

A string is a constant, ie it is immutable (it cannot be modified). For to modify a string it is obligatory to create a new one. It can be delimited either by single quotation marks ('), or by double quotation marks (").

```
s 1 =    ' This is a thong
s 2 =    " This is also a thong
```

Single and double quotes are equivalent, but can be used to introduce a string in a string, as in the example below:

```
>>> print('This is a string" within a string')
This is a string" within a string
```

4.1.1 Operations

As for the assignment of a number, the assignment of a string to a variable is done with the equal symbol (=):

```
str=    ' This is a thong
```

The concatenation of two strings, to create a single string containing the two elements, is done with the plus symbol (+):

```
>>> str=    'Hello    +    'world! '
>>> str
'Bonjour Monde ! '
```

You can create a string that contains several repetitions of another string with the symbol star (*):

```
>>> str = ' Hello ' ÿ 3
>>> str
' Hello Hello hello '
```

4.1.2 Construction of a string with the percentage symbol (%)

If the symbol % appears in a string followed by one of the letters d, f, s, e or x, this indicates that we want to replace this symbol by a variable of our script. It will therefore be necessary to follow the string of another symbol %, then of the variable or variables that we wish to introduce in the thong. If more than one variable is to be specified, enclose them in parentheses.

The letters after the symbol % mean that we want to introduce:

d = an integer
f = a real number
s = a string
e = a number in exponential format
x = a number in hexadecimal format

```
>>> has , b , c = 1 , 3 , 7
>>> print ( '%d 1 3 , %de and %d are prime numbers ' % ( a , b , vs ) )
and 7 are prime numbers
>>>
>>> has , b , c = 1.0 2 , 3 . , 7. 6 7
>>> print ( '%f 1. 0 , 2 %and %fare s real numbers ' % ( a , b , vs ) )
2 0 0 0 0 , 3. 2 0 0 0 0 0 and 7. 6 7 0 0 0 0 are real numbers
>>>
>>> has , b , c = ' Bill' , 'Bob' , ' Joe '
>>> print ( '%s %and %sare probably American ' % ( a , b , vs ) )
Bill , Bob and Joe are probably American
```

Note that real numbers have 6 decimal values by default. We can modify this number by inserting a point, then the number of decimal places that we want, between the sign % and the letter f:

```
>>> has , b , c = 1. 0 2 1 2 , 3 . 2 , 7. 6 7
>>> print ( '%.2 f 1. 0 , %.2 fet %.2 f are s real numbers ' % ( a , b , vs ) )
2 3. 2 0 and 7. 6 7 numbers
```

This can also be applied to indicate with how many digits we want to represent a integer or to restrict the number of characters possible when inserting a digit into a string.

4.1.3 Some methods of the string class

upper(): converts all characters to uppercase

```
>>> a = ' hello '
>>> a = a . upper ( )
>>> has
 'HELLO'
```

lower(): converts all characters to lowercase

```
>>> a = 'HELLO'
>>> a = a . ower ( )
>>> has
 ' Hello '
```

capitalize(): Make sure the first letter is uppercase

```
>>> a = ' hello '
>>> a = a . capitalize ( )
>>> has
 ' Hello '
```

replace(old, new): replaces all occurrences of old with new.

```
>>> a = ' hello '
>>> a = a . replace('l' , 'b')
>>> has
 'abo'
```

find(sub): returns the index of the first occurrence of sub, returns -1 if sub does not exist.

```
>>> a = ' Good day all >>> the world ! '
a . find('') day 3
```

join(seq): returns a string that contains the concatenation of all elements of seq. The element separator is the string on which the method is applied.

```
>>> a = ' Hello '
>>> b = ' ' . join(a)
>>> b
 ' Hello '
```

split(sep): returns a list of words contained in the string using sep as a separator.

```
>>> a = ' Good day all >>> the world ! '
b = a . split('')
>>> b
 [ ' Hello ' , ' all ' , ' the ' , 'world ! ' ]
```

For the details of the methods of the string class, refer to the documentation from Python.

4.1.4 Indication

We can easily obtain a character or a character string contained in a string with the indication method. As for all elements considered as sequences (string, list, tuple), we access the elements of the sequence by specifying between square brackets the index that we want to obtain. A single value will return the character at that position in the string, while 2 colon separated values (e.g. [2:8]) will return a string containing the specified substring:

```
>>> a = ' Hello everybody ! '
>>> a [ 0 ]
'B'
>>> a [ 5 ]
'u'
>>> a [ 2 : 1 0 ]
ndayto >>>
a [ : 1 0 ] # from start until aposition 10
Good day to
>>> a [ 1 6 : ] # of position 16 in the world! >>> a [ -1] ' at the end
# lastposition
' ! '
>>> a [: -4] # from start to end minus last 4 positions
Good day everyone
```

4.1.5 The len method

len(string): the len() function returns the number of characters, including empty spaces, in a string :

```
>>> a = ' Hello everybody ! ' >>> len(a)
22
```

4.1.6 Documentation String

A string in triple quotes (' ' will be ' or """) on the line following the declaration of a function stored in the function's doc variable. --

```
>>> defp ui ssance ( x , exp=2):
... ' ' ' C al c ul eandreturning the value of xalap ui ssance exp returnxyy exp ...
...
...
>>> power . document --
' C al c ul e and return the al eur of xalap ui ssance exp '
```

4.2 Lists: Operations on lists

A list simply consists of a sequence of values separated by commas and placed in parantheses [ ]. The elements in a list can be of different types (number, string, list, tuple, dictionary, function, object, ...) and can be mixed. The list is alterable, that is to say that we can modify its elements without having to create a new list.

```
>>> a = [ 3 >>> , 5 , 2 , 8 , 7 , 5 , 1 ]
a . sort ( )
>>> has
 [ 1 , 2 , 3 , 5 , 5 , 7 , 8 ]
```

4.2.1 Some operations on lists

— list1 + list2: concatenation of list1 and list2

— list[i]: returns the element at position i

— list[i : j]: returns a list containing the elements between positions i and j

— len(list): returns the length of the list

del list[i]: delete the element at position i

— list.append(val): appends the element val at the end of the list

— list.extend(list): add a list at the end of a list

— list.sort(): puts the elements of a list in order

— list.reverse(): reverses the elements of a list

— list.insert(i, val): inserts val at position i

— list.count(val): returns the number of occurrences of val in a list

— list.pop(): returns and eliminates the last value of a list

— val in list: returns True if the element val is present in the list

Very good tutorial About lists in the Python documentation:

4.3 The range function

The range function generates an integer iterator which can be very useful for creating quickly a list of integers (Under python 2, the range function directly returns a list). It allows, among other things, to loop over a certain number of values inside a loop for.

```
>>> list ( row ( 5 ) )
 [ 0 4,] , 2 , 3 ,
1 >>> foriinr an ge ( 5 ):
 . . .  print( i end=, ” ” )
 . . .
0 1 2 3 4
```

We will elaborate on building repetitive instructions in the next lesson.

Arguments of the range function

If only one argument is given to the range function, it will be taken as the maximum value and the list will start at 0. If two arguments are given to the function, they represent the value minimum and the maximum value. Which will generate a list of the minimum (inclusive) value to the largest possible value smaller than the maximum value. A third argument can be given to specify the 'step', i.e. how many values to advance at each element from the list.

```
>>> list ( row ( 2 [ 2 3 4 7 ] ,   1 0 ) )
>>> list ( row ( 2 5 [ 2 , 8 6 ] ,   ,  8 9 , ]
                            ,  1 0 ,  2 ) )
    ,  4 6 ,    ,
>>> list ( range ( 1 0 [ 1 0    ,  5 0 ,  5 ) )
     ,  1 5 ,  2 0 ,  2 5 ,  30 ,  3 5 4 , 0    ,  4 5]
```

4.4 Practice

4.4.1 Controlling objects via initialization parameters

We have just seen how to modify the behavior of the range function by giving it arguments between parentheses. The behavior of objects from the pyo library can also be governed in this way. However, there are a few rules to follow:

1 - If the names of the arguments are not specified, they must be given in order.

Following this method, considering that the order of the arguments of the Sine object is:

freq, phase, mul, add

To assign an amplitude value of 0.5 (mul argument), one must specify all the arguments which precedes it:

```
a = If ne ( 1 0 0 0    ,  0 ,  . 5 )
```

2 - Some arguments can be omitted or the order can be mixed only if we specify the name of the arguments by assigning them a value.

```
a = If ne ( mul=.5)
b = If ne ( mul=.3       ,  freq=500)
```

3 - We can mix the 2 methods, but as soon as an argument is duly named, the order is considered to be broken and we must specify the name for the rest of the arguments.

```
a = Si ne ( mul=.5) b
= Harmonizer ( a , ÿ12, mul=.3 , feedback =0.5)
```

The manual therefore becomes an indispensable tool when writing a script. You can view the manual online if you are connected to the web or use Python's built-in help system by calling the help method:

```
>>> from pyo import ÿ >>> h
el p ( If ne )

Help on class If nein module py oli b . generators:
 . . .
```

The pyo library also provides a class args function allowing to quickly obtain the initialization parameters of an object as well as their default value:

```
>>> from pyo import ÿ >>>
classargs ( If ne )
 ' If ne ( freq=1000, phase=0, mul=1, add=0) '
```

A good habit is to always have an interpreter open when writing a script!
As a bonus, you can listen to the examples in the manual with the example function:

```
>>> from pyo import ÿ >>>
example ( If not )

import time
from pyo import ÿ s = S
erver ( ) . boot()s. start ( )
sine = If ne ( freq =500) .
out() time. sleep(5.0 0 0 0 0 0) sec.
stop() time. sleep(0.25)s. shutdown ( )
```

4.4.2 Play a sound file on the hard disk

We are now going to experiment with the SfPlayer object which allows us to play a sound file saved on the hard disk.

```
>>> from pyo import ÿ >>>
classargs ( S fPl ayer )
 'S fPlayer ( path , speed=1, loop=F al se          ,   offset=0, interp=2, mul=1, add=0) '
```

The path argument is required since no value is assigned to it by default. The path to the sound is specified using a string using the syntax used on Unix systems, i.e. by separating the elements hi erarchical by the symbol / .

”/Users/olivier/Desktop/snds/baseballmajeur s.aif”

```
>>> from pyo import ÿ >>>
s = S erver(). boot() >>> s.
start ( ) >>> a = S fPl ayer
( path=”/ U se rs / olivier /Desktop/ snd s / baseb allm ajeurs .aif”) ). out ( )
```

On the Windows platform, python will take care of converting your string into the correct format before fetching the sound from disk. We would therefore write:

”C:/Documents and Settings/olivier/Desktop/snds/baseballmajeur s.aif”

The optional parameters control respectively:

— speed: The playback speed (0.5 = twice as slow, 2 = twice as fast) — loop: whether the sound is looped or not (True = looped, False = played once) — offset : The starting point, in seconds, in the sound file (2 = playback starts at 2 seconds in the file) — interp: The quality of the interpolation on playback (don't touch it for now!) — mul: The amplitude factor (0.5 = twice as strong) — add : The value to add to each of the samples after the reading (Do not modify in the case of a sound file reading as this may damage your speakers and your ears!)

Exercise

Play a sound in a loop and apply a chain of sound processing chosen from those introduced in the previous lesson. Manage effect parameters via initialization arguments.

4.4.3 Polyphony management

If in the previous exercise you loaded stereo sound, you may have noticed that signal is being sent to both speakers. This is due to the fact that each object of the pyo library is able to manage several audio streams at the same time. A stream is an internal library object responsible for conveying a monophonic sound channel. Each object manages its audio via

one or more streams, which are picked up by the server and routed to a sound card output. The len function, applicable to lists, can also be called with a pyo object as argument. It will return the number of streams handled by the object.

```
>>> from pyo import ÿ >>> s
= S erver(). boot() >>> s. start
( ) >>> a = S fPl ayer ( path="/
U se rs / olivier /Desktop/ snd s / baseb allm ajeurs .aif") ). out() >>> len(a) 2
```

When it comes to distributing an object's streams to the audio outputs, the default behavior (modifiable via out method arguments) is to alternate the output, depending on the number of available outputs. , with each new stream. Starting at 0, a stereo sound will be distributed on outputs 0 and 1, ie the first stream on the left and the second on the right.

Arguments of the out method

- chnl: The channel where the first stream of the object will be sent.
- inc: How many channels we advance with each new stream.
- dur: Duration of activation, in seconds, of the object. The stop method is called automatically when time is up. A duration of 0 means an infinite duration.
- delay: Delay, in seconds, before the activation of the object.

In a multi-channel environment, these parameters can be manipulated in very subtle ways. In stereo, the default values (chnl=0, inc=1) are generally the most consistent!
You can convert the number of channels of an object with the mix method as follows:

```
>>> a = If ne ( )
>>> len ( a ) 1

>>> b = a . mix(2)
>>> len(b) 2

>>> print(b)
< Instance of Mix class >
>>> c=S fPl ayer("/U se rs/olivier/Desktop/snd s/baseb allm ajeurs.aif") >>> len(c)2

>>> d = c . mix(1)
>>> len(d) 1

>>> print(d)
< Instance of Mix class >
```

Note that b and d are new objects of class Mix that were created by calling the method mix. They receive the signals from a and c, perform the conversion and manage new streams... a and c remain intact.

4.4.4 Using lists for musical purposes

Each object of the pyo library takes a certain number of arguments in order to define its behaviour. For example, the path parameter of the SfPlayer object will take a link to a sound file on the hard disk. Recall the line to play a sound from the hard disk:

```
a = S fPl ayer ( ”/home/ olivier /Dropbox/private/snd s/baseb allm ajeurs.aif” )
```

A very powerful concept in programming, called multi-channel expansion, consists of the possibility of using a list as the value of an argument. It is a concept generalized to all the parameters of the pyo objects (unless otherwise mentioned in the manual!), in order to multiply the sound processes without having to repeat the same lines of code over and over again. Thus, to create a gradual delay in the playback of a sound, we would execute the following code.

```
1 from pyo import ÿ
2s = Server ( ) . boot()
3 # In a path , the symbol ”                    . .”    moved back a folder in the hierarchy.
4 # A relative path starts at the folder where the python script is stored.
5 a = S fPl ayer ( path=” . . / snd s / baseb allm ajeurs . aif ” , . 9 9 2 ] ,                ,
6                         speed = [ 1    , 1 . 0 0 5 , 1 . 0 0 7 loop=True , mul(=).25) .
7 sec. gui( locals ())
```

scripts/chapter 04/01 snd_chorus.py

The range function, seen a little earlier, could be used to create a sequence of harmonics:

```
1 from pyo import ÿ
2s = Server ( ) . boot()
3 a = Si ne ( freq=list ( r an ge ( 1 0 0 mul = [ .    , 1 0 0 0 , 1 0 0 ) ) ,
4                 5 , . 3 5 , . 2 g ul (4 locals(s) )       , . 0 7 , . 0 5 , . 0 3    , . 0 1 ] ) . out ( )
```

scripts/chapter 04/02 harmonic_series.py

An object that receives a list as the value of one of its arguments will create the number of streams necessary in order to calculate the desired sound reproduction. If, for the same object, several arguments receive a list as input, the longest list will be used to generate the number of streams required. Values from shorter lists will be used in order, with looping when the end of the list has been reached. Using lists of different lengths for the arguments of a object will produce a rich and varied sonic result since the combinations of parameters will be different for each instance of the sound process.

```
1 from pyo import ÿ
2s = Server ( ) . boot()
3 a = FM( carrier =[ 5 0 india x    , 7 4. 6 , 1 0 1 , 1 2 5. 5 ]  ,  ratio = [ 1 . 5 0 1    , 2 . 0 2 , 1 . 0 0 5 ] ,
4                 =[ 6 4 5 print ( ren    , 7 , 6 , 5 , 9 , 6 5 8 ,  , 5 ( 6 ] , mul=.05) . out
( a ) )
6 sec. gui( locals ())
```

scripts/chapter 04/03 fm_stack.py

The lists given to the arguments of the FM object being of different lengths, the largest is used to determine the number of streams of the object, i.e. 12 (length of the list given to the index argument). By looping over the values of the smallest lists, we get 12 frequency modulations with as parameters:

	FM carrier	ratio	index
1	50	1.501	6
2	74.6	2.02	5
3	101	1.005	125.5 4
4	1.501		7
5	50	2.02	6
6	74.6	1.005	5
7	101	1.501	9
8	125.5	2.02	6
9	50	1.005	5
10	74.6	1.501	8
11	101	2.02	5
12	125.5	1.005	6

Distribution of the values to the arguments of the different frequency modulations.

Caution

A pyo object being considered as a list, it is very easy to create an overload of calculation for the processor by asking to create several instances of an already expensive effect (a reverb for example) on a signal. The solution will be to pass a mix of all the signals (streams) from source to effect. Let’s take as an example our shifted playback (4 stereo playbacks):

```
>>> a=S fPl ayer(”../snd s/baseb allm ajors.aif” speed=[1 1.0 0 7 ,
                                                    , 1 . 0 0 5 ,  , . 9 9 2 ] ,  loop=True , mul=.25)
>>> len(a)
8
>>> b = Free rb(a). out ( )
>>> len ( b )
8
```

CPU: 20%

```
>>> a=S fPl ayer(”../snd s/baseb allm ajors.aif” speed=[1 1.0 0 7 ,
                                                    , 1 . 0 0 5 ,  , . 9 9 2 ] ,  loop=True , mul=.25)
>>> len(a)
8
>>> b = Free rb(a.mix(2)). out ( )
>>> len ( b )
2
```

CPU: 16%

4.4.5 Examples and exercises

Exercises

1. Write a script playing 5 sine waves of increasing frequencies {400, 500, 600, 700, 800} and decreasing amplitudes {.5, .25, .12, .06, .03}. The 5 sine waves must be generated with a single line of code!
2. Resume the exercises of the previous course (processing chains from sources synthesis) and eliminate the ctrl() method by giving arguments to the initialization of objects. Generate multiple tones with a single processing chain simply by modifying the initialization parameters (see example 1 below).
3. Magnifying the sounds of exercise number 2 using the potential of lists as arguments (see example 2 below).
4. Create a processing chain on a sound file playback. Test with different sounds and different values to the arguments of the objects.

Examples

Example number 1: Specifying arguments when initializing objects.

```
"""
S pecification of the arguments to the initialization of sobjects .

Three sounds based on example no. 3 from the previous lesson:

Example no 3: C onnec ti on s in parallel (multiÿsources)
        lfo 1 ÿ>
        lfo 2 ÿ> ha rm onize r
        lfo 3 ÿ>

Example #1:
        C on son ant tuning with a slight frequency vi a ti on
        to create a modulation effect.
Example #2:
        F requencies of the LFOs very close to the tha rm o ni sa ti on of
        1/10th semitone with feedback to create a flange effect.
Example #3:
        Total disagreement , sonoriteser approaching the noise.
"""

from pyo import ÿ

s = Server ( ). boot()

example = 1 # modify this variable to change example

if example == 1:
        src 1 = LFO( freq =100, sh a rp =.5 , mul=.1) . out ( )
        src 2 = LFO( freq=150.5, mul=.1) . out ( ) =.25
        src 3 = LFO( freq =200.78 , mul=,1) . out ( ) =.15
        harm = Harmonizer ( src 1+src 2+src 3 transpo=ÿ7) . out ( 1 )
elif example == 2:
```

```
        src 1 = LFO( freq =100, sh a rp =.75 , mul=.1) . out ( )
        src 2 = LFO( freq =99.8 sh a rp =.75 , mul=.1) . out ( )
        src 3 = LFO( freq =100.3 sh a rp =.75 , mul=.1) . out ( )
        harm = Harmonizer ( src 1+src 2+src 3 transpo =0.1 feedback =.8 , mul=.6) . out ( 1 )

        src 1 = LFO( freq =100, sh a rp =.75 , mul=.1) . out ( )
        src 2 = LFO( freq =123, sh a rp =.65 , mul=.1) . out ( )
        src 3 = LFO( freq =178, sh a rp =.5 , mul=.1) .out ( )
        harm = Harmonizer ( src 1+src 2+src 3 transpose =2.33 , feedback =.5) . out ( 1 )

sec. gui( locals ())
```

scripts/chapter 04/04 arguments.py

Example number 2: Enlargement of tones using lists as arguments.

```
"""

Extension of sonorites with the help of lists in argument,
over the base of stroissonorites of the previous example.

Example #1:
        Chorus on each source + multiple sha rm o ni sa ti ons.
Example #2:
        F requencies of the LFOs very close to 3 frequencies ,
            legeresha rm o ni sa ti ons with feedback to create a
        and flange effect.
Example #3:
        Total disagreement , even more pr uspr oche noise.
"""

from pyo import *

s = S erver () . boot()

example = 1 # modify this variable to change example

if example == 1:
        s 1 = LFO( freq = [ 1 0 0 s 2 , 1 0 0 . 0 3 , 9 9 . 9 5 , 9 9 . 9 1 ] , mul=.05) . out ( )
        = LFO( freq = [ 1 5 0 . 4 1 s 3 = , 1 5 0 . 5 , 1 5 0 . 7 8 , 1 5 0 . 2 ] , mul=.05) . out ( )
        LFO( freq = [ 2 0 0 . 7 8 h = , 2 0 1 , 2 0 1 . 3 , mul=.05) . out ( )
        Harmonizer ( s 1+s 2+s3 , transpo =[ ÿ12 ÿ7 ,5 , 7] , mul=.1 )
elif example == 2:
        s 1 = LFO( freq =[ 1 0 0 s 2 , 3 0 0 , 5 0 0] , mul=.1 ) . out ( )
        = LFO( freq =[ 9 9. 8 s 3 = , 3 0 0 . 3 , = [ . 1 , . 0 1 5] ) . out ( )
        LFO( freq =[ 1 0 0. 3 h = , 2 9 9. 7 6 , 4 9 9 , mul=.1 ) . out ( )
        Harmonizer ( s 1+s 2+s3 , feedback=.8 , mul=.4) . out ( 1 )
elif example == 3:
        s 1 = LFO( freq =[ 1 0 0 s 2 , 2 7 6 , 4 2 1 , 5 1 1] , mul = [ . 1 , . 0 1 ] ) . out ( )
        = LFO( freq =[ 1 2 3 s 3 = , 3 2 4 , 3 8 9 , 4 8 mul = [ . 1 , . 0 1 ] ) . out ( )
        LFO( freq =[ 1 7 8 h = , 2 6 5 , 4 4 4 , 8] , 5 2 sh a rp =.5 , mul = [ . 1 , . 0 1 ] ) . out ( )
        Harmonizer ( s 1+s 2+s3 , 6 1] , transpo =[ ÿ3. 7 6 , 2 2. 3 3] , feedback =.5) .

sec. gui( locals ())
```

scripts/chapter 04/05 list in args.py

Chapter 5

Loops and importing modules

5.1 Importing modules

It is impossible to automatically import all functions available in Python as soon as the software is opened, because there are virtually an infinity of them. Built-in functions represent those that are likely to be used most often. All other functions are grouped by themes in .py files that can be imported as modules (pyo is a module). To use the functions contained in a module, it is absolutely necessary import this module. There are two forms of import in Python.

First form: importing a complete module

```
import sys, print   time
(sys.path)
print (time.time())
```

The path attribute of the sys module, as well as the time function of the time module are then accessible. Since it is the module itself that has been imported, you must call the attribute or the function by first specifying the name of the module (module.attribute or module.function()).

Second form: importing elements from a module

```
from random import r andin t print ( r    ,   uni form
andin t ( 1 0 0 2 0 0 ) )     ,
print (uniform(0 1))      ,
```

Here, only the randint and uniform functions from the random module have been imported. They can therefore be used without specifying the name of the module to which they belong.

Machine Translated by Google

Import all functions of a module

```
from random import ÿ
```

The star means 'all'!

Modify reference name

You can also modify the reference name of a module (mainly to lighten the syntax) with the keyword as. In the example below, all the functions of the wx.html module can be called with the syntax html.fonction().

```
import wx . html as html

win=html. HtmlWindow()
```

The dir(module) function returns the list of attributes and functions belonging to a module.

```
>>> import math
>>> dir (math)
[ 'doc , _ _ file _ _ , _ _name _ _ , acos , a if , atan , atan2 , ceil' ,
 cos , cosh' , 'degrees' , , , exp' modf ' 'fabs , n' floor, , 'fmod' , 'frexpsin ,
 'hypot' , , 'lde xp , log, , ' log 1 , , ft 'pow' , ra di years , ,
 if nh ' , sqrt , tan , 0 'tanh ' ]
>>>
```

5.2 Repetitive instructions (while, for)

We have already explored 2 of the 3 main types of control structures in programming, the sequence and the conditional statement. We will now elaborate on the third type, repetition. Two methods for programming loops will be presented here, the while instruction, allowing to create loops having an indefinite duration, and the instruction for, allowing to cycle through all the elements of a group.

5.2.1 The while statement

Let's take a simple example:

```
a = 0
w hil e a < 1 0:
    # statement block
    print (a)
    a += 1
# continuation of the program
```

A variable a is first initialized. Then the while statement is called followed by a condition (don't forget the colons which indicate the beginning of a statement block). while, which means while , means that as long as the condition is met, the following block of instructions (note the indentation) must be repeated in a loop. Let's see in detail:

1. the while statement evaluates the condition. If it is false, the whole block is skipped and the program jumps to the line following the block. If the condition is true, the program executes the instructions of the block:
2. the value of variable a is displayed.
3. the variable a is incremented by 1.
4. The while statement is then re-evaluated based on the new value of variable a.
   If the condition is still true, the block is executed again, until the condition is false.

It is therefore very important, in order not to create infinite loops, that there is at least one variable of the condition which is modified inside the block of instructions of the loop. Under certain conditions, infinite loops can be defined in a program. In this case, a user action, or a command sent when exiting the program, terminates the loop.

If a variable is used in the condition of a while statement, it must be
be set before executing this statement.

Boolean values True and False can be used to create an infinite loop, or to
more common, to activate or deactivate a block of code in a program.

```
while True :
        # The condition is always true. . . if F al se: # These
instructions will never be executed if True: # R eac ti
        vele bl oc d ' instructions                                    ,   replace with :
```

Creation of a table of squares and cubes of the numbers from 1 to 10:

```
>>> a = 0
>>> w hil ea < 1 0: a +=
 . . .          1 print ( a
 . . .                    , a ÿ ÿ2  , a ÿ ÿ3 )
 . . .
1 1 1
2 4 8
3 9 27
4 16 64
5 25 125
6 36 216
7 49 343
8 64 512
9 81 729
10 100 1000
```

We are now going to create a program allowing to display the numbers of the sequence of Fibonacci (where a value is the sum of the 2 previous values). The display will stop when a certain threshold (defined by the user) will be reached.

```
1 threshold = 100
2 a = b = 1
3 w hil e b < threshold:
4         print (bb=b ,   end=" " )
5         at ,              ,   a+b
6 print ( )
```

scripts/chapter 05/01 fibonacci.py

This program will return the following result:

```
1 2 3 5 8 13 21 34 55 89
```

There are several interesting aspects in this little program:

1. Multiple assignments allowing to give the value 1 to variables a and b simultaneously.
2. Note the end argument given to the print function. It eliminates the inserted carriage return automatically by the print function (by inserting a space instead), thus allowing to display several values on the same line.
3. Parallel assignments inside the loop (a, b = b, a+b). In one fell swoop, we modifies the state of the two variables (a and b).

5.2.2 The for statement

The for statement is used to cycle through all the members of a group, usually a list. Each member will be assigned, in turn, to a variable that can be used inside the loop. The syntax of a for loop is very simple:

```
>>> foriprint(i    inrange ( 5 ):
 . . .            end=" " )    ,
 . . .
0 1 2 3 4
```

The range function, with a single argument, returns an iterator generating a list containing values from 0 to the argument value - 1. If two arguments are present, they will represent the start value and the end value (not included) of the list. A third argument can be used to define the step , i.e. the value of the increment between each member of the list.

```
>>> list ( row ( 5 ) )
 [ 0 4,]     ,  2 ,  3 ,
1 >>> row ( ( 5 1 0 ),)
 [ 5 9,]  6 ,  7 ,  8 ,
>>> range ( ( 0 [ 0   , 1 0 , 2 ) )
     ,  2 ,  4 ,  6 ,  8 ]
```

The for loop is very useful to apply a transformation on all the values of a listing :

```
>>> list 1 = [ 2 >>>     ,4 ,6 ,8 ,1 0 ]
list 2 = [ ]
>>> foriinlist 1:
 . . .            # ' i ' takes the value of each element of the list around its role
 . . .             listing 2 . append ( i ÿ ÿ2 )
 . . .
>>> print ( list 2 )
 [ 4 1,1 6 3 6     ,6 4 ,
```

The method append is part of the methods of the class list and allows to add a value at the end of the list on which the method is applied. We use a method using the syntax object.method().

Variant of the previous program, using the length of the list:

```
>>> list 1 = [ 2 >>>     ,4 ,6 ,8 ,1 0 ]
foriinr an ge ( len ( list 1 ) ):
 . . .            # ' i is an index x in listlist 1 [ i ] = list 1 [ i ] ÿ ÿ 2     ,   0 ÿ> len ( list 1 ) ÿ 1
 . . .
 . . .
>>> print ( list 1 )
 [ 4 1,6 3 6 1 0 0 ],6 4 ,
```

len is a Python built-in function that returns the number of elements contained in a group (list, tuple, dictionary, character string) given as an argument. This variant constitutes what is called a destructive action on an assembly (in-place modification), the list values are directly replaced without resorting to a second list.

5.3 List generators ('list comprehension')

List building is one of the most powerful features of the Python language. She allows to create complex lists very quickly by a particular use of the loop for. List comprehensions are delimited by square brackets and consist of an expression followed by a for loop, with the possibility of nesting several for loops or if expressions.

```
[    express if we      ' loop key      condition (optional) '          ]
```

— The square brackets indicate the creation of a new list.

— expression is the statement block of the for loop (usually an expression mathematical).

— loop is the repetitive for statement allowing to execute the expression a certain number of times.

— condition is an optional parameter allowing to specify under which conditions the expression must be executed.

Simple generator

```
>>> malist = [ ifori >>> malist inrange ( 2 0 ) ]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>>
>>> squares = [ i ÿ ifori inrange ( 5 ) ]
>>> squares
[0, 1, 4, 9, 16]
```

Generator with nested for loops

```
>>> malist = [ [ i , j ] fori >>> malist inr an ge ( 3 ) forj inrange ( 3 ) ]
[[00][0 , , 1], [0, 2], [1, 0], [1, 1], [1, 2], [2, 0], [2, 1], [2, 2]]
```

In this example, for each loop turn of variable i, the loop of variable j is performed three times. The long version of the creation of this list would be written as follows:

```
mylist = [ ]
for i inrange (3):
    for j inrange (3):
        my list . append([i,j])
print (mylist)
```

Generator with a condition

In the case of a conditional generator, the expression will be executed (and the result will be added to the list) only for loop turns where the condition is met.

```
>>> malist = [ [ i , j ] fori >>> malist inr an ge ( 3 ) forj inrange ( 3 ) ifj != 1 ]
[[00][0 , , 2], [1, 0], [1, 2], [2, 0], [2, 2]]
```

Long version :

```
mylist = [ ]
for i inrange (3):
    for j inrange (3):
        if j != 1:
            my list . append([i,j])
print (mylist)
```

Execution time

One of the great advantages of list generators is the optimization of the underlying code which noticeably improves the performance of the program. A small example that demonstrates the time required to create a list containing one million integers:

```
import time

num = 1000000

# “list comprehension” t=time.
time() l = [i for i print (time.time()
ÿ t) #0. 1 9 3 0 9 7 1 n4 5 6 (num) ]
sec.

”
# loop ” for t =
time . time ( ) l = [ ]
for i inr an ge (num) :
l . append(i) print (time. time() ÿ t) #0.
2 9 9 5 8 0 0 9 7 1 9 8
sec.

# always use a t = time function. time() l = list ” built-in ”
( r an ge (num)) print (time. time() ÿ t) #0.0 4 0
6 4 2 0 2 3 0 8 6 5 sec.
```

5.4 Practice

5.4.1 Using the for loop to generate parameters

Some examples of using the for loop to facilitate the construction of lists of parameters.

Generation of an additive synthesis by the addition of sine waves whose frequencies are integer multiples of a fundamental frequency.

The unavoidable :

```
from pyo import ÿ s =
Server ( ) . boot()
```

Creation of empty lists where the values assigned to the frequency and amplitude parameters will be added:

```
freqs = [ ]
amps = [ ]
```

We define a fundamental frequency:

```
bottom = 50
```

`

Inside a for loop, we will fill our lists by applying simple functions mathematics on the variable i (which will take in turn the values from 1 to 20):

```
for i      inrange ( 1 2 1 ,):
        freqs. append ( i ÿ bottom )
        amps. append(0.35/i)
```

Display of filled lists:

```
                                    '
print ( '     F requencies:      ' ,   freqs )
print ( ' Amplitudes:              , amps )
```

By giving our lists to the parameters of a Sine object, we create as many oscillators as there are there are values in the lists:

```
a = If ne ( freq=freqs , mul=amps ) . out ( )
```

Another essential:

```
s. gui( locals ())
```

Questions :

— What must be done to obtain a different fundamental frequency?

— What should be done to add or remove harmonics?

```
1 # E xer ci cebou cl e 2       ” for      ”,   series of frequencies and amplitudes.
from pyo import ÿ
3
4s = Server ( ) . boot()
5
6 freqs = [ ]
7 amps = [ ]
8
9 background = 50
10    for i inrange ( 1 2 1 ):    ,
11          freqs. append ( i ÿ bottom )
12          amps. append(0.35/i)
13                                    '
14 print ( '       F requencies:      ' ,   freqs )
15 print ( ' Amplitudes:                , amps )
16
17 a = If ne ( freq=freqs , mul=amps ) . out ( )
18
19 sec. gui( locals ())
```

scripts/chapter 05/02 loop for 1.py

Modification of the previous example in order to give it a little more naturalness...

The main defect of this type of algorithm is the excessive precision of the values. Any its naturalness cannot produce such just harmonics and it is precisely in the tiny variations as the sound comes to life. We must therefore find a way to detune the harmonics relative to each other. To do this, we will rely on “chance” as well only to the python random module, the chance generator par excellence!

Let's start by importing the random module:

```
random import
```

We will then modify our for loop to include a slight variation in the frequency of each harmonic:

```
        dev=random. uni form (0 . 9 8 freqs . ,  1 . 0 2 )
        append ( i ÿ fond ÿ dev )
```

The uniform function of the random module requires 2 parameters which are respectively the minimum possible value and the maximum possible value. Each time the function is called, a new value is picked from the allowed range. We multiply the frequency of the oscillator by this value to create a slight deviation.

Experiment with different deflection ranges to see the impact on sound!

```
 """
1
2 E xer ci cebou cl e       " for  ",  version 2    ,  series of frequencies and amplitudes.
3  """
4
5 from pyo import ÿ
6 import random
7
8s = Server ( ). boot()
9
10 freqs = [ ]
11 amps = [ ]
12
13 background = 50
14 for i inr an ge ( 1 dev =    ,2 1 ):
15          random . uni form ( . 9 8 freqs .      , 1 . 0 2 )
16          append ( i ÿ fond ÿ dev )
17          amps. append(0.35/i)
18                                  '
19 print ( '      F requencies: 20   ', ,  freqs )
print ( ' Amplitudes:                 , amps )
21
22 a = If ne ( freq=freqs , mul=amps ) . out ( )
23
24 sec. gui( locals ())
```

scripts/chapter 05/03 loop for 2.py –

Synthesis by frequency modulation using some of the functions offered by the random module.

The following script uses controlled random distribution generators, accessible via the random module, to create a different frequency modulation synthesis sound each time the script is executed.

— random.triangular(min, max): Triangular distribution (favors median values) between min and max.

• random.choice(seq): Picks a random value from the list given to the seq parameter.

— random.randint(min, max): Picks a random integer between min and max.

Consult the Python manual for a detailed description of the functions of the random module.

```
"""
E xer ci cebou cl e " for ", synthesis by frequency modulation.
"""

from pyo import ÿ
import random

s = Server ( ). boot()

char = [ ]
rats = [ ]
ind = [ ]
for i inr an ge ( 1 0 ):
    because . append ( random . triangular ( 1 5 0 , 1 5 5 ) )
    rat . append ( random . ch oi ce ( [ . 2 5 , . 5 , 1 , 1. 2 5 , 1 . 5 , 2 ] ) )
    ( random . ran di nt ( 2 , 6 ) )

print ( ' Carrier: ', because )
print ( ' Ra tio: ', rat)
print ( ' Index: ', ind )

fm = FM( carrier=car , ratio=rat , india x=ind , mul=.05) . out ( )

sec. gui( locals ())
```

scripts/chapter 05/04 loop for FM.py

5.4.2 List generators ("list comprehension")

Sometimes the operations necessary to generate a list do not require several lines of code. It is then possible to insert the expression even into a list generator. This technique allows for example to create a huge chorus in a single line:

```
a = FM( [ random . uni form ( 2 4 0 , 2 6 0 ) for i inr an ge ( 2 0 0 ) ] , mul=.005) . out ( )
```

200 FM synths were created by this line.

One could reproduce the example of synthesis by frequency modulation using the list generators:

```
from pyo import ÿ
import random

s = Server ( ) . boot()

fm = FM( carrier =[random . triangular ( 1 5 0 , 1 5 5 ) for i inr an ge ( 1 0 ) ] 1 1. ,
ratio=[random. ch oi ce ( [ . 2 5 inde , . 5 , 2 5 1 . 5 , , 2 ] ) for i inr an ge ( 1 0 ) ] , ,
x =[random . ran di nt ( 2 , 6 ) for i inr an ge ( 1 0 ) ] mul=.05) . out ( )

sec. gui( locals ())
```

scripts/chapter 05/05 listComp FM.py

Exercises

1 - In the following script, you must replace the 3 empty lines (...) so that your script generates a chorus of 10 voices (slightly different speeds) by playing a sound loop on the hard drive. You must create a mono mix from the chorus and then apply a 3-part harmonization including the lower octave (-12), the upper third (4) and the upper fifth (7). Pay attention to the amplitudes! The distribution of votes is done as following :

Left: The original chorus and the upper fifth.
Right: The lower octave and the upper third.

```
from pyo import ÿ
from random import uni form
s = Server ( ) . boot()

. . .
. . .
. . .

s. gui( locals ())
```

2.1 - Knowing that a square wave is only composed of odd harmonics whose amplitudes correspond to the inverse of their rank, which list generators would would make it possible to obtain the frequencies and the corresponding amplitudes of such a sound?

For a fundamental of 100 Hz, we want the following components:

100 Hz, amplitude = 1
300 Hz, amplitude = 1/3.
500 Hz, amplitude = 1/5.
700 Hz, amplitude = 1/7.
...

2.2 - Transform your list generators in order to create a chorus of this note...!

Solutions on page 68.

5.4.3 Continuous variations of parameters using pyo objects

A natural sound is not static, it is usually constantly changing. Whether through slight variations in frequency or amplitude, the timbre of a sound varies over time.
We will now start using pyo objects for the purpose of creating control paths, i.e. audio signals not meant to be sent to the speakers but rather to create variations of parameters over time. Its use is very simple. If the manual specifies that an argument accept floats or PyoObjects, then it is permissible to pass it a previously created pyo object. To give a slight frequency variation to an oscillator, the following script creates an object that generates continuous random values (Randi) that we assign to the freq argument of an oscillator:

```
1 from pyo import ÿ 2 s = S
erver ( ) . boot()
3
4 rnd = Randi( min=390, max=410, freq =4) 5 a = If ne
( freq=rnd , mul=.5) . out ( )
6
7 sec. gui( locals ())
```

scripts/chapter 05/09 randi.py

4 times per second, Randi picks a new value between 390 and 410 and linearly interpolates from the current value to the picked value.

Generating a chorus note

It should be remembered that a pyo object is considered as a list (the length of the list corresponds to the number of audio streams it contains). Being considered as a list, if we assign to an argument an audio object of length x, the newly created object will also contain x audio streams. It is a double-edged sword! It is very easy to saturate the processor by multiplying the calculations, but it is also very easy to build complex sounds in a few lines. Here is an example of a chorused note:

```
1 from pyo import ÿ 2 import
random
3
4s = Server ( ) . boot()
5
6 fr = Randi( min=295, max=300, freq =[random .uni form(2 7 sines =  ,8 ) for i  inrange ( 1 0 0 ) ] )
SineLoop (frfeedback=.08, mul=.01).out() 8 print ( len (sines))


9
10 sec. gui( locals ())
```

scripts/chapter 05/10 var_freqs.py

The Randi object, receiving a list of 100 values at the freq argument, will create 100 variations random. By assigning this object to the freq parameter of an oscillator, this one will automatically generate 100 oscillators of different frequencies. Pay attention to the amplitude!

A similar example, this time the variations are applied to the amplitude of 10 oscillators:

```
1 from pyo import ÿ
2 import random
3
4s = Server ( ) . boot()
5
6 amp = Randi( min=0, max=.05 7 sines =      ,    freq=[random. uni form ( . 2 5              , 1 ) for i inr an ge ( 1 0 ) ] )
SineLoop ( freq =[ i ÿ100+50 for i inr an ge ( 1 0 ) ]                                          ,    feedback=.03 , mul=amp) . out ( )
8
9 sec. gui( locals ())
```

scripts/chapter 05/11 var_amps.py

Exercises

1. For the previous two examples, test different values to the arguments min, max and freq of the Randi objects in order to show the impact of each on the sound result.
2. Replace Randi objects with Randh objects, what is the difference?

Generation of predetermined random numbers

It is sometimes desirable to be able to predict, to some extent, the content of a generation random. The generation of a harmony is a case in point where the frequency values do not cannot simply be randomized. The Choice object allows us to specify, at using a list, the values that can come out of the freelance. Here is a small example, with 4 voices of polyphony, where frequencies are restricted to notes in the C major scale:

```
1 from pyo import ÿ
2
3s = Server ( ) . boot()
4
5 pits = [ midiToHz (m) for min [ 3 6 6 choice = Choice  , 4 3 , 4 8 , 5 5 , 6 0 , 6 2 , 6 4 , 6 5 , 6 7 , 6 9 , 7 1 , 7 2]]
( choice=pi tsfreq =[ 1 , 4] )                          ,                  , 2 , 3
7 sines = SineLoop ( freq=choice , feedback =.05 , mul=.1) . out ( )
8
9 sec. gui( locals ())
```

scripts/chapter 05/12 chime.py

The midiToHz function converts a Midi note value (an integer between 0 and 127) into the corresponding frequency in Hertz.

5.4.4 Examples and exercises

Exercises

1. Repeat the previous exercises and replace the sound sources to test different sounds.
2. Apply random generators to the various source control parameters. Exploring objects in the randoms category of pyo. * Attention * It is from here that it is dangerous for your ears! Check below volume if your automations do not cause too large dynamic gaps.
3. Complete processing chains by adding effects. The categories filters, effects and dynamics offer several signal transformation processes.

Examples

The example below is based on 4 voice polyphony, with random frequency picking on the C scale of the previous script.

```
pits = [ midiToHz (m) for min [ 3 6 ch oi x = Choice ,4 3 ,4 8 ,5 5 ,6 0 ,6 2 ,6 4 ,6 5 ,6 7 ,6 9 ,7 1 ,7 2] ]
( ch oi ce=pi tsfreq =[ 1 , 4] ) , ,2 ,3
sines = SineLoop (freq=choice, feedback=.05, mul=.1) . out ( )
```

First, let's duplicate the source using a frequency modulation (FM) synthesis generator. As a bonus, a little portamento has been added to changes of frequencies to eliminate potential clicks.

```
pits = [ midiToHz (m) for min [ 3 6 ch oi x = Choice ,4 3 ,4 8 ,5 5 ,6 0 ,6 2 ,6 4 ,6 5 ,6 7 ,6 9 ,7 1 ,7 2] ]
( ch oi ce=pi tsfreq =[ 1 # P e ti tt ruc a portamento ,2 ,3 , 4] )
frequency changes eliminating clicks!
chport = Port( choice sines = , risetime=.001 , falltime=.001)
SineLoop ( freq=chport # Second s or FM , feedback=.05 , mul=.1) . out ( )
synthesis rce ,
fms = FM( carrier=chportratio =1.0025 , , india x=4, mul=.025) . out ( )
```

Let's add two automations on the feedback of the SineLoop and on the index of the modulation of frequency. Both LFOs are phase reversed to alternate the predominance of one source relative to the other.

```
pits = [ midiToHz (m) for min [ 3 6 ch oi x = Choice ,4 3 ,4 8 ,5 5 ,6 0 ,6 2 ,6 4 ,6 5 ,6 7 ,6 9 ,7 1 ,7 2] ]
( ch oi ce=pi tsfreq =[ 1 chport = Port ( choice risetime ,2 ,3 , 4] )
=.001 # LFO, in phase , lffeed = Si ne Sine( freq =0.1 sines , falltime=.001)
( freq=chport # LFO, out of phase, SineLoop feedback FM index
, mul=.07 , add=.07)
, feedback=lffeed , mul=.1) . out ( )

lfind = If ne ( freq =0.1 , phase =0.5 , mul=3, add=3)
fms = FM( carrier=chport india x=lfindration =1.0025 , , mul=.025) . out ( )
```

Now let's add a first effect, a flange effect, created using the delay line whose delay time varies over time. First of all, don't forget to mix the stereo audio streams if needed (we have 4 voice polyphony) and sum sources. Then we add the delay with a sine LFO to the delay parameter, which will have the effect scanning of the spectrum by a comb filter. Note that the out() methods have been removed from SineLoop and FM objects to act only on the sum of the two sources.

```
pits = [ midiToHz (m) for min [ 3 6 ch oi x =          ,4 3 ,4 8 ,5 5 ,6 0 ,6 2 ,6 4 ,6 5 ,6 7 ,6 9 ,7 1 ,7 2]]
Choice ( ch oi ce=pi tsfreq =[ 1 chport =, Port( choice      ,2 ,3 , 4] )
risetime =.001 lffeed = Si ne ( freq =0.1 sines =      , falltime=.001)
=chport , mul=.1)                    ,  mul=.07     ,  add=.07)
                                        -          ,  feedback=lffeed
lfind = If ne ( freq =0.1 , phase =0.5 , mul=3, add=3)
fms = FM( carrier=chportratio =1.0025 india x=lfind # Addition of the stereo    , mul=.025)
mix of two sources
s rc sum = sines . mix(2) + fms. mixing ( 2 )
# Delay with LFO on the delay time to create a flange effect
lfdel = If ne ( .1 add =.005)          mul=.003     ,
comb = Delay (src sum feedback =.5, d el ay=lfdel) #    ,
Send sum to speakers
s rc sum . out ( )
```

Finally, we direct the signal to a reverb which will assume the balance between the signal original and reverberated signal as well as sends it to the speakers. Here is the final version of our program :

```
1 from pyo import ÿ
2
3s = Server ( ) . boot()
4
5 pits = [ midiToHz (m) for min [ 3 6                  ,4 3 ,4 8 ,5 5 ,6 0 ,6 2 ,6 4 ,6 5 ,6 7 ,6 9 ,7 1 ,7 2]]
6
7 ch oi x = Choice ( ch oi ce=pi tsfreq =[ 1 8, chport =      ,2 ,3 , 4] )
Port( choice risetime =.001          ,                    ,  falltime=.001)
9
10 lffeed = Si ne ( freq =0.1 sines =    ,  mul=.07     ,  add=.07)
11   SineLoop ( freq=chport              -          ,  feedback=lffeed          , mul=.1)
12
13 lfind = If ne ( freq =0.1 , mul=3, add=3), phase =0.5 14
fms = FM( carrier=chportratio =1.0025 india x=lfind              ,                    , mul=.025)
15
16s rc sum = sines . mix(2) + fms. mixing ( 2 )
17
18 lfdel = If ne ( . 1 19       ,  mul=.003     ,  add=.005)
comb = Delay ( src sum     -       ,  d el ay=lfdelfeedback=.5)
20
21 out sum = s rc sum + comb # Sum of sources and delay
22 # Send to verb and signal output
23 re v = WGVerb( out sum -       ,  feedback=.8      ,  cutoff=3500, bal=.4) . out ( )
24
25 sec. gui( locals ())
```

scripts/chapter 05/13 example final.py

5.4.5 Solutions to exercises on 'lists comprehension'

```
#! /usr/bin/env python
# enc odin g: utf ÿ8

S ol u ti on for exercise 1 on the 'list comprehension' .

from pyo import ÿ
from random import uniform

s = S erver () . boot()

sf = S fPlayer (SNDS PATH+' / transparent . aif ' speed =[ uni , loop=True , mul=.07 ,
form ( . 9 9 , 1 . 0 1 ) for i inr an ge ( 1 0 ) ] ) . mixing ( 1 )
sf. out ( )
harm = Harmonizer ( sf , transpose =[ ÿ1 2 , 7 , 4] , mul=.4) . out ( 1 )

sec. gui( locals ())
```

scripts/chapter 05/06 ex 1 listComp.py

```
#! /usr/bin/env python
# enc odin g: utf ÿ8

S ol u ti on for Exercise 2. 1 on 'list comprehension' .

from pyo import ÿ
from random import uniform

s = S erver () . boot()

max harms = 20
freqs = [ i ÿ 100 for i inr an ge ( max harms ) amps = [ 0 . 3 if ( i %2) == 1 ]
5 / i for i inr an ge ( max harms ), print ( "F requency s : freqs ) if ( i %2) == 1 ]
" ,
print ( "Amplitudes: , amps )
a = If ne ( freq=freqs , mul=amps ) . mix(1). out ( )

sec. gui( locals ())
```

scripts/chapter 05/07 ex 2 1 listComp.py

```
1 #! /usr/bin/env python
2 # enc oding: utf ÿ8
3
4 S ol u ti on for exercise 2. 2 surles                    'list comprehension'          .
5   """"
6
7 from pyo import ÿ
8 from random import uniform
9
10s = S erver () . boot()
11
12 max harms = 20
13
14 freqs = [ i ÿ 100 ÿ uni form ( . 9 9                    , 1 . 0 1 )
15                 for i inr an ge ( max harms ) for j              inrange ( 1 0 )      if ( i %2) == 1 ]
16
17 amps = [ 0 . 0 3 5 / i for i              inr an ge ( max harms ) for j inr an ge ( 1 0 ) if ( i %2) == 1 ]
18
19 print ( "Frequency s: 20 print "  ,   freqs )
( "Amplitudes:                    , amps )
21
22 a = If ne ( freq=freqs , mul=amps ) . mix(1). out ( )
23
24 sec. gui( locals ())
```

scripts/chapter 05/08 ex 2 2 listComp.py

Chapter 6

Functions

This course elaborates on the creation of custom functions and the use of these inside a script.

6.1 Simple function without parameter

Creating a function consists of grouping together a certain number of operations, linked together, performing a particular process. A previously defined function can be called anywhere in a script, allowing programs to be written compactly and efficiently. The group of operations may consist of only a few rows or be a complex grouping of operations spanning several hundred rows. Here is a simple example of a function without parameters and its use in a script:

```
from math import sqrt

def squareroot_2 4 ( ) : print_( '
        squareroot of 2 : %f ' % sqrt ( 2 ) ) print ( ' squareroot of 3 : %f ' % sqrt ( 3 ) )
        print ( ' squareroot of 4 : %f ' % sqrt(4))


squareroot 2 4 ( )  _  _
```

Note the creation of a statement block with the colon ending the header line and block indentation.

This function only prints the square roots of the numbers 2 through 4, but it already demonstrates the basic syntax needed to create a function. This always begins with the keyword def followed by the name of the function and parentheses (we will see later that they are not always empty). The colon ends the header line, and all subsequent indented lines make up the block of statements that will be executed each time the function is called. To complete the construction of the function, it suffices to return to the main line of indentation.

6.2 Function with parameters

When creating a function, it is possible to specify parameters by assigning them an argument between parentheses. The arguments will take the values given when calling of the function and can be used as normal variables in the execution of the block instructions. This allows to apply a sequence of operations on different values simply by varying the parameters given to the call.

```
from math import sqrt

def squareroot(x):
    print ( ' square root of %f: %f ' % ( x , sqrt(x)))

squareroot ( 2 5 )
squareroot ( 1 2 )
```

If more than one parameter is necessary for the proper execution of the function, it is sufficient to separate with a comma between the parentheses.

```
from math import sqrt

def squareroot (mini , max):
    print ( ' the square roots of s number between %d and %d are: ' % ( mini for i inr an ge ( mini , max ) )
    maxi ):
        print (sqrt(i), end=" " )

squareroots ( 5 , 1 0 )
```

6.3 Default Parameter Values

A particularly interesting feature is to give default values to the parameters of a function. This allows some parameters to be omitted when calling without create runtime errors. A function can therefore be created giving a lot of latitude to the user while allowing him to use it in simpler variants. When creating the function, a default value is assigned to a parameter by following the argument corresponding to the sign "=" then to the desired value.

```
def power ( x , exp=2):
    print ( '%.2 f exp os an t %.2 f gives: %.2 f ' % ( x , exp , pow( x ,exp ) ) )

power ( 5 )
power ( 2 1 6 ) ,
power ( 2 , exp=32)
power ( x=3, exp=2)
power ( exp=3, x=2)
```

We can see that there are several different ways to call this function. One can specify only the parameter with no default value, or specify both values (the value given to the argument exp overrides the default value). It is also possible to refer directly to the variable whose value you want to specify with the syntax variable name=value.

A few rules to follow

1. When defining a function, parameters with no default value must precede parameters with default value.
2. When calling, if the names of the variables are not referred to, the values must be specified in order.
3. During the call, one cannot define a parameter by specifying the name of the variable and then assign a value to a parameter without explicitly specifying its name. When a variable name is given, the order of the parameters is no longer respected, so the interpreter will not know which parameter to assign the value to.

6.4 Local variables and global variables

At this point, it is important to understand how namespaces are shared in Python. Namespaces are registers where variables specific to certain environments are stored, for example the different functions and classes. In order to avoid conflicts between variables with the same name, the interpreter traverses the namespaces in a precise order to identify the different variables. As a general rule, the interpreter searches in the most local space possible, and if it does not find the requested variable, it goes back up the floors to the space global names. We will have the opportunity to come back to this and delve a little deeper into this subject when we talk about classes and objects. For the moment, two types of variables are of particular interest to us, local variables and global variables.

A global variable is a variable defined at the first level of indentation of a script, thus accessible anywhere in the script, while a local variable is a variable defined inside inside a function and is only accessible within the function itself. It is therefore quite possible that a variable initialized at the beginning of a script has a name x, and that a variable with the same name is defined inside a function, without creating a conflict. However, you have to be aware of the namespace where you are in order to know what the real value of the variable you are using is. Small example:

```
>>> a = 1
>>> deff() : a = 2
 . . .
 . . .
>>> f ( )
>>> print ( a ) 1
```

In this example, a variable a is initialized to the value 1, then a function, giving the value 2 to a variable a, is called. Then, the value of the variable a is displayed and we

find that it is 1, not 2, even though the last operation performed on a variable a was an assignment of the value 2. The reason is that these two variables are not part of the same namespace, the former is part of the script's global namespace, while the latter is part of the function's local namespace. They are therefore two separate variables.

```
>>> a = 1
>>> deff() : a = 2
 . . .
 . . .          print(a)
 . . .
>>> f ( ) 2

>>> print(a) 1
```

Although the variable defined in the function is not accessible from the outside, the opposite is quite possible. That is, it is allowed to access the value of a variable belonging to the global namespace inside a function, without being able to modify it. .. for the moment.

```
>>> a = 1
>>> deff(): print(a)
 . . .
 . . .
>>> f ( ) 1
```

If one wishes to act on the content of a global variable inside a function, it must be declared as a 'global variable' inside the function itself. This declaration is done with the global keyword and tells the interpreter not to create a new local variable but to use, and modify, the global variable.

```
>>> a = 1
>>> deff():
 . . .          globalaa =
 . . .           2
 . . .
>>> f ( )
>>> print ( a ) 2
```

6.5 Using functions with value return

The main purpose of a function is to perform operations on parameters and provide a result. This result can be returned by the function and stored in another variable, which can be used as is in the rest of the program. When the return keyword is encountered in a function, it terminates the function and returns the value(s)

who follow him, if there are any. We can assign, with the symbol = a , the return of a function to variable. This operation is illustrated in the following excerpt.

```
>>> a = 0
>>> def increment(x):
 . . .              returnx + 1
 . . .
>>> a = increment ( a )
>>> print(a)
 1
>>> a = increment ( a )
>>> print(a)
 2
```

The return of a function can be a value of any type, for example, a list:

```
a = 16

def powerlist_2 ( exp ):
        exponents = [ ]
        for          inrange ( 1 , exp+1):
            iexhibitors . append ( i ÿ ÿ2 )
        return exhibitors

myList = listpower 2 (_a )
```

6.6 Practice

6.6.1 Using functions in a script

We will now explore the interaction with the audio signal of a script using previously defined functions. As a first example, we will define 3 sound sources (a sound, an FM synthesis and an additive synthesis) which will be activated in turn by calling of a function. The three sources:

```
1 from pyo import ÿ
2 import random
3s = Server ( ) . boot()
4
5 # play a sound file
6 snd= S fPl ayer (SNDS PATH+ "/accord.aif" , speed=[.9 9 8 loop=True,              , 1 . 0 0 3 ] ,
7                                 mul=.35) . stop ( )
8 # FM synthesis
9 fm = FM( carrier = [ 9 9 10 #     , 9 9 . 7 , 1 0 0 , 1 0 0 . 4 , 1 0 0 . 9 ] ,   ratio=.4987       ,   india x=8, mul=.1) . stop ( )
syntheseadditive
11 adsyn = SineLoop (freq=[random.uni form(1 4 5, mul=.05).stop(1 5 5 ) for i       inrange ( 2 0 ) ]      ,
12                                 feedback=.15
13
14 sec. gui( locals ())
```

scripts/chapter 06/01 interaction 1.py

Note the call of the stop method on each of the sources. This will have the effect of not start calculating the samples automatically (the play method is almost always called as soon as an object is created).

Using text-input in the server window

Now that we know about Python's namespaces, we can clear up the argument to the gui method of the Server object. locals is a built-in function of Python which returns all variables defined so far in the current namespace (the script namespace in most cases). Thus, all the variables defined before server window call are known from the input-text, which is no more and no less than a interpreter extension. It is therefore possible to launch new commands from this tools. This will be used to modify the state of certain objects or to launch functions previously defined. For example, to start the calculation of the FM synthesis, we will write, followed by the key enter:

```
f.m. out ( )
```

Let's add a function to our script allowing to start a source while stopping the one that was playing previously. An argument will be used to determine which source should play. This function is inserted after the definition of the sources and before the display of the server window.

```
1 from pyo import ÿ
2 import random
3s = Server ( ) . boot()
4
5 # play a sound file
6 snd= S fPl ayer (SNDS PATH+ ”/accord.aif” , speed=[.9 9 8 loop=True, , 1 . 0 0 3 ] ,
7                  mul=.35) . stop ( )
8 # FM synthesis
9 fm = FM( carrier = [ 9 9 10 #  ,9 9 . 7 , 1 0 0 , 1 0 0 . 4 , 1 0 0 . 9 ] ,  ratio=.4987  ,  india x=8, mul=.1) . stop ( )
syntheseadditive
11 adsyn = SineLoop (freq=[random.uni form(1 4 5, mul=.05).stop() 1 5 5 ) for i  inrange (20)],
12                   feedback=.15
13
14 # src ÿ> { ' snd ' 15 def  ,  'fm'  ,  'adsyn'}
play ( sr c=”snd” ):
16      if src ==  'snd'  :
17          nd . out ( )
18          f.m. stop ( )
19          adsyn. stop ( )
20      elif src == 'fm' snd .  :
21          stop ( )
22          f.m. out ( )
23          adsyn. stop ( )
24      elif src == 'adsyn' snd .  :
25          stop ( )
26          f.m. stop ( )
27          adsyn. out ( )
28
29 sec. gui( locals ())
```

scripts/chapter 06/02 interaction 2.py

In order to smooth the input and output of sound in our program, we are going to apply a amplitude envelope to each of our sources. The Fader object allows you to set a fade time rise (triggered by the play method) and a fall time (triggered by the stop method) an amplitude control that will be assigned to the mul argument of the source. No need to call the stop method on Fader objects, this is one of the rare cases where the play method is not called automatically when the object is created. This time we'll let the sources play in permanently and only control the amplitudes via the control function.

```
from pyo import ÿ
import random
s = Server ( ) . boot()

snd amp = Fader (fad ei n=5, fadeout=5, hard=0)
snd = S fPl ayer (SNDS PATH + "/accord .aif" , speed = [ .9 9 8 .3 5 ) . out ( ) , 1 . 0 0 3 ] ,
                  loop=True , mul=snd amp ÿ
fm amp = Fader (fad ei n=5, fadeout=5, hard=0)
fm = FM( carrier = [ 9 9 1 0 0 . 4 , 1 0 0 . 7 ] , ratio=.4987 , india x =8,
          mul=fm amp ÿ
adsyn amp = Fader (fad ei n=5, fadeout=5, dur=0)
adsyn = SineLoop (freq=[random.uni form(1 4 5 1 5 5) for i inr an ge (2 0)] .0 5). out ( ) ,
                  feedback =.15 , mul=adsyn amp ÿ

# src ÿ> { 'snd' def , 'fm' , 'adsyn'}
play (src):
    if src == snd 'snd' :
        amp . play ( )
        fm amp. stop ( )
        adsyn amp . stop ( )
    elif src == 'fm' snd :
        amp . stop ( )
        fm amp. play ( )
        adsyn amp . stop ( )
    elif src == 'adsyn' snd :
        amp . stop ( )
        fm amp. stop ( )
        adsyn amp . play ( )

sec. gui( locals ())
```

scripts/chapter 06/03 interaction 3.py

6.6.2 Automatic call of a function with the Pattern object

The process of the Pattern object is to call a function cyclically. The Argument function takes the name of a previously defined function (note that there are no parentheses, this is a reference and not a call) and the argument time , the time elapsed between each call. Here is how we will create this object:

```
pat = P a t te rn ( func ti on=play , time=6) . play ( )
```

The play function will be called every 6 seconds, with no parameters. We will therefore have to modify the latter in order to eliminate the parameter which points to a source. We will rather

create a mechanism that picks up a random source:

```
last = None
def play():
    global last
    src=random. ch oi ce ( [ ' snd ' w hil e  ,  'fm'  ,  'adsyn' ] )
    src == last :
        src=random. ch oi ce ( [ last  'snd'  ,  'fm'  ,  'adsyn' ] )
    = src
    if src ==  'snd'  :
        . . . .
```

Several things to note here. First, after choosing a random string from the list of possible choices (don't forget to import the random module), we set a condition in a while loop to ensure that the draw is not identical to the previous one. The loop will run a new trap until src is not equal to last. Then we replace the content of the global variable last by the new trap to prepare the next condition.

```
1 from pyo import ÿ
2 import random
3s = Server ( ) . boot()
4
5 snd amp = Fader (fad ei n=5, fadeout=5, hard=0)
6 snd = S fPl ayer (SNDS PATH + "/ acc o rd . aif " , speed = [ .9 9 8 mul=snd  , 1 . 0 0 3 ] ,
7                   loop=True , 8  amp ÿ .3 5 ) . out ( )
fm amp = Fader (fad ei n=5, fadeout=5, hard=0)
9 fm = FM( carrier = [ 8 0 8 1 . 4 1 7 9 7 8 1 ,  adsyn amp =  , 8 1 . 9 ] ,  ratio=.4987  ,  india x =8, mul=fm amp ÿ . 1). out ( )
Fader ( fad ei n =5, fadeout =5, dur=0)
11 adsyn = SineLoop (freq=[random.uni form(1 4 5 1 5 5) for i inr an ge (2 0)],
12                   feedback =.15 , mul=adsyn amp ÿ . 0 5 ). out ( )
13 last = None
14 def play ( ):
15     global  last
16     src = random . ch oi ce ( [ ' snd ' w hil  ,  'fm'  ,  'adsyn' ] )
17     e src == last :
18         print ( '"%s " ' %src) al ready pl a yi ng , choo si ng an o the r . . .
19         src=random. ch oi ce ( [ ' adsyn' ] )  'snd' 'fm'  ,
20     last=src
21     if src ==  'snd'  :
22         snd amp . play ( )
23         fm amp. stop ( )
24         adsyn amp . stop ( )
25     elif src == 'fm' snd  :
26         amp . stop ( )
27         fm amp. play ( )
28         adsyn amp . stop ( )
29     elif src == 'adsyn' snd  :
30         amp . stop ( )
31         fm amp. stop ( )
32         adsyn amp . play ( )
33
34 pat = P a t te rn ( func ti on=play , time=6) . play ( )
35 sec. gui( locals ())
```

scripts/chapter 06/04 interaction 4.py

Variables defined inside a function are automatically destroyed when the execution of the function is finished, so it is necessary to declare the variable last at the first script indentation level. This way it will be available for the lifetime of the program. However, since we want to modify it inside the function, we must declare it as a global variable. If the global last line is omitted, a new variable local to the function will be created and will no longer be available on the next call.

6.6.3 Generating a compound sequence (Score object)

The Score object is used to organize the unfolding of a series of events over time.
The input argument expects an audio signal, containing integers, which can be generated either with a Counter object to create a sequential sequence, or with a RandInt object to create a dice. random rolling. A second argument (fname) specifies the generic name of the functions that will be called. When Score receives a new integer, the function with the name given in fname, with the suffix of the integer in question, is called. Here is a script that illustrates the structure of a sequential organization:

```
1 from pyo import ÿ 2 s =
S erver ( ) . boot()
3
4 def event 0 ( ):
5         print ( ”Call function 0” )
6
7 def event 1 ( ):
8         print ( ”Call to function 1” )
9
10 def event 2 ( ):
11         print ( ”Call of function 2” )
12
13 def event 3 ( ):
14         print ( ”Call of function 3” )
15
16 met = Metro (time=1) . play ( ) 17
count = Counter (met , min=0, max=4) 18 score
= Score (count fname =” event” )
19
20 sec. gui( locals ())
```

scripts/chapter 06/05 score.py

Variation, with the Score object, of the exercise previously created with the Pattern object

Let's go back to our 3 sources from the previous exercise and, this time, let's organize the events in time. We will first create an independent function for controlling each of the sources. We will take advantage of this to slip in a small element of chance each time a source should start. If the function is called with the value 1 as argument (default value), the sound plays, otherwise it stops:

```
def pl aysnd (state=1):
    if state == 1:
        nd . speed = [ random . ch oi ce ( [ . 5 , . 6 7 , . 7 5 snd amp . , 1 , 1 . 2 5 , 1 . 5 ] ) for i inrange ( 2 ) ]
        play ( )
    else :
        snd amp . stop ( )

def play fm (state=1):
    if state == 1:
        freq = random . ran di nt ( 0 7 ) ÿ25+50
        f.m. carrier = [ freq ÿ random . uni form ( . 9 9 fm amp . play , 1 . 0 1 ) for i inrange ( 5 ) ]
        ( )
    else :
        fm amp. stop ( )

def pl ayadsyn (state=1):
    if state == 1:
        freq = random . ran di nt ( 0 7 ) ÿ25+50
        adsyn. freq = [ freq ÿ random . uni form ( . 9 9 adsyn amp . , 1 . 0 1 ) for i inrange ( 2 0 ) ]
        play ( )
    else :
        adsyn amp . stop ( )
```

We then define the functions that will be called in turn, which will constitute our sequence of events:

```
def event 0 ( ):
    play fm ( )
def event 1 ( ):
    pass
def event 2 ( ):
    pl aysnd ( )
def event 3 ( ):
    pla ayadsyn ( )
    play fm ( 0 )
def event 4 ( ):
    pass
def event 5 ( ):
    pl aysnd ( 0 )
    play fm ( )
def event 6 ( ):
    pla ayadsyn ( 0 )
def event 7 ( ):
    pl aysnd ( )
def event 8 ( ):
    play fm ( 0 )
    pla ayadsyn ( )
def event 9 ( ):
    pass
def event 1 0 ( ):
    pl aysnd ( 0 )
    pla ayadsyn ( 0 )
    put . stop ( )
```

Finally, a metronome sends clicks to a Counter object, which advances by 1 with each click again, and the count is given to a Score object which calls the various functions to Alternatively.

```
met = Metro (time=5) . play ( )
count = Counter (met , min=0, max=11)
score = Score ( count " )        ,   fname=" event
```

In the next lesson, we will elaborate in more detail on the concept of "trigger", the audio signal which is at the heart of the synchronicity of events in pyo. This signal is generated, among other things, by the Metro object.

```
from pyo import ÿ
import random

s = Server ( ) . boot()

snd amp = Fader (fad ei n=5, fadeout=5, hard=0)
snd = S fPl ayer (SNDS PATH + "/accord .aif" , speed = [ .9 9 8 .3 5 ) . out ( )                , 1 . 0 0 3 ] ,
                                loop=True , 9      mul=snd amp ÿ
fm amp = Fader (fad ei n=5, fadeout=5, dur=0)
fm = FM( carrier = [ 9 9 1 0 0 . 4   , 9 9 . 7 , 1 0 0 ,          , 1 0 0 .   ,   ratio=.4987        ,
                                india x =8, mul=fm amp ÿ 12      9 ] . 1). out ( )
adsyn amp = Fader ( fad ei n =5, fadeout =5, dur=0)
adsyn = SineLoop (freq=[random.uni form(1 4 5 1 5 5) for i inr an ge (2 0)] feedback=.15.0 5). out ( )            ,
                                                        , mul=adsyn amp ÿ

def pl aysnd (state=1):
        if state == 1:
                nd . speed = [ random . ch oi ce ( [ . 5 , . 6 7 , . 7 5 snd amp . , 1 , 1 . 2 5 , 1 . 5 ] ) for i          inrange ( 2 ) ]
                play ( )
        else :
                snd amp . stop ( )

def pl ay fm (state=1):
        if state == 1:
                freq = random . ran di nt ( 0 7 ) ÿ25+50
                f.m. carrier = [ freq ÿ random . uni form ( . 9 9 fm amp . play  , 1 . 0 1 ) for i          inrange ( 5 ) ]
                ( )
        else :
                fm amp. stop ( )

def pl ayadsyn (state=1):
        if state == 1:
                freq = random . ran di nt ( 0 7 ) ÿ25+50
                adsyn. freq = [ freq ÿ random . uni form ( . 9 9 adsyn amp .   , 1 . 0 1 ) for i          inrange ( 2 0 ) ]
                play ( )
        else :
                adsyn amp . stop ( )

def event 0 ( ):
        play fm ( )
def event 1 ( ):
        pass
```

```
def event 2 ( ):
        pl aysnd ( )
def event 3 ( ):
        pla ayadsyn ( )
        play fm ( 0 )
def event 4 ( ):
        pass
def event 5 ( ):
        pl aysnd ( 0 )
        play fm ( )
def event 6 ( ):
        pla ayadsyn ( 0 )
def event 7 ( ):
        pl aysnd ( )
def event 8 ( ):
        play fm ( 0 )
        pla ayadsyn ( )
def event 9 ( ):
        pass
def event 1 0 ( ):
        pl aysnd ( 0 )
        pla ayadsyn ( 0 )
        put . stop ( )

met = Metro (time=5) . play ( )
count = Counter (met , min=0, max=11)
score = Score ( count " )        ,   fname=" event

sec. gui( locals ())
```

scripts/chapter 06/06 score audio.py

Bonus: small example of the use of the Pattern object controlling a bank of filters.

```
from pyo import ÿ
import random

s = Server ( ) . boot()

amp = Fader ( fad ei n =0.25 noz = N ,   fadeout=0.25 , hard=0, mul=0.3) . play ( )
oi se ( mul=amp)
fbank = ButBP( noz freq =[ 2 5 0                    , 7 0 0 , 1 8 0 0 , 3 0 0 0] , q=40, mul=4) . out ( )

def change ( ):
        f 1 = random . uniform ( 2 0 0 5 0 0 ) ,
        f 2 = random . uniform ( 5 0 0 1 0 0 0 )
        f 3 = random . uniform ( 1 0 0 0 2 0 0 0 )
        f 4 = random . uniform ( 2 0 0 0 4 0 0 0 )
        fbank. freq = [ f1 f2 f3 f 4 ],      ,      ,

lfo = If ne ( .1 add =.75),   mul=.5    ,
pat = P a t te rn ( func ti on=change , time=lfo ) . play ( )

sec. gui( locals ())
```

scripts/chapter 06/07 pattern filters.py

Chapter 7

Directory management and review

7.1 Reading and writing files in Python

The open function returns an object with which to read or write a file on disk.
Two arguments are required, the first is a string indicating the name of the file to read or write, while the second indicates the mode, i.e. the way in which we want to interact with the file. If the file to be read is in the current directory, it can be called simply by its name, otherwise, the full link must be given. The current directory is the one in which the 'python' command was launched and can be found with the getcwd method of the os module (os.getcwd()). The possible modes, identified by a string, are:

— 'r': Open a read-only file.

— 'w': Open a file for writing only (If a file of the same name already exists, it will be erased).

— 'a': Opens a file in append mode. The new data will be added after the data already present.

— 'r+': Opens a file for reading and writing.

On Windows and Macintosh systems, if a 'b' is appended to the mode, the file will be opened in binary format. Which gives the modes 'rb', 'wb' and 'r+b'.

7.1.1 File object methods

The read method returns a string containing all the text contained in the file. If the read is already rendered at the end of the file, the read method returns an empty string.

```
>>> f = open('my file. txt_>>> f. read()      ,     r  ' )
                                                                                    '
 ' This is the first line of the . \ nAnd this is the second wave! \n >>> f . read again ( )
 ..
>>>
```

** Note the carriage return symbol. **

```
\not
```

The readline method reads and returns one line of the file at a time. As for the method read, if the read is rendered at the end of the file, an empty string will be returned.

```
>>> f = open('my file. txt >>> f. readline()
' This is the first line of the file. \ not
>>> f . readline()
'And this is the second wave! \n >>>
f . readline()

>>>
```

It is possible to read all the lines of a file and store them as a list with the readlines method.

```
>>> f = open('my file. txt >>> f. readlines()
[' This is the first line of the . \ not ', 'And this is the second wave! \ not ']
>>>
```

The write method writes lines to a file.

```
>>> f = open('my file. txt >>> f. w ri te('H
ell o world!\n')
>>>
```

When the operations on a file are finished, it is necessary to close the file with the method closed.

```
>>> f = open('my file. txt >>> f. write('H
ell o world!\n >>> f. close()
>>> f = open('my file. txt >>> str = f. read()

>>> str
' Bonjour Monde ! \n'
>>> f . close()
```

7.1.2 Concrete example of writing and reading a text file.

The following example illustrates the saving of a note sequence, generated using a algorithm, in a text file and playing it back for musical purposes. The advantage of save the sequence in a text file is that if the generated sequence is interesting, it is always accessible in text format, even after exiting the program.

```
1 ################################################ ############################
2 # ÿÿÿ Writingandreadingfilesontheharddisk. #
3 # 1 ÿ Genera ti on of 4 melodies of 16 notes each in a . #
4 #2 ÿ Rereading the file and converting lines into listed 'integers. #
5 # 3 ÿ Little algo that loops over notelists and controls a synth. #
6 ############################################## ##########################
7 from pyo import ÿ
8 import random
9
10s = S erver () . boot()
11
12 # C minor harmonic (on pi gelesnotes in this list 'ai ded' un inde x).
13 scale = [ 3 6 3 8 4 ,8 5 6 8, 3 9, 4 1 , 4 3 , 4 4 , 4 7 , 5 0 , 5 1 , 5 3 , 5 5 , 5 9 , 6 0 , 6 2 , 6 3 , 6 5 , 6 7 , ,7 1 , 7 2]
14 #Reference to the length of the list ÿ1 because ran di nint includes
maximum .
15 maxn = len ( scale ) ÿ 1
16 # Starting index for the genera ti on of sm el o di es .
17 index x = random . ran di nt ( 0 , maxn )
18
19 # Random walk function (returns +/ÿ 3 with respect to the previous index).
20 def drunk ( ):
21     # We want to modify the global variable " inde x global inde x ", and not create a local one.
22
23     # We add di ti onnea " inde x inde " an integer randomly drawn between ÿ3 and 3 .
24     x += random . ran di nt ( ÿ3 ,3)
25     # Ensure that inde x is always between 0 and maxn ( length of list ) .
26     if inde x < 0:
27         india x = abs ( india x )
28     elif inde x > maxn:
29         india x = maxn ÿ ( india x ÿ maxn )
30
31 ### Writing a text file ###
32
33 # Open a file in write mode. "
34 f = open ( " "w" ) ratings. txt ,
35 # Generates 4 rows of 16 values.
36 for i inr inrange ( 4 ):
37     for ge # ( 1 6 ):
38         Change globalvariable drunk ( ) " india x ".
39
40         # Write the new index followed by a space . w ri te ( str ( x , in the file .
41         index ) + " " )
42     # After each group of 16 values f . write( "\n" ) , we change lines.
43
44
45 # Close the file
46f . close()
47
```

```
48 ### Reading a textfile ###
49
50 # Open the file in read mode and retrieve the m el o di es .
51 f= open ( ”          ratings. txt          ”,    ”r ” )
52 # Each line is an element (a string of 16 digits) of a list.
53 mellist=f. readlines()
54 # We close the file.
55f . close()
56
57 ### C onve r si on from a string of characters to listed integers ###
58
59 # We e nl evelescarriage returns .
60 mellist = [ l . replace( ”\n” , ”” ) for l in mellist]
61 # We separate the data in each list.
62 mellist = [ l . split ( ) for l in mellist ]
63 # We convert each of sstrings to integer .
64 for i in range ( 4 ):
65         for j inrange ( 1 6 ):
66             mellist[i][j] = int (mellist[i][j])
67
68 # I here      ,   mellistcon holds 4 lists of 16 indexes to read in the list                    ” scale ” .
69
70 ############## Audi o process ( synth ol dscho ol ) ################
71
72 # Amplitude Envelope
73     f = Fader ( fad ei n =.005 fadeout =.05 hard =.125 , mul=.1)
74 # 10 canned frequencies (to initialize a chorus)
75 freqs = [ 1 0 0] ÿ 1 0
76 #10 square waves.
77 a = LFO( freq=freqs , type=2, mul=fa) sh rp =.75
78 # synth ÿ> distortion ÿ> reverb
79 pha = Di sto (a.mix(2), 80 re v =      d rive =.85     ,   sl ope =.95     , mul=.2)
WGVerb( pha, feedback=.75                    ,   cutoff =3000, b al =.2) . out ( )
81
82 ############## S ec ti oncontrols ################
83
84 # Global V a ri a bl es . . .
85 mel = 0 # Each of 4 melodies.
86 count = 0 # Count of s beats in the melody.
87 def new note ( ):
88         global mel #     ,   count
89         Get ind xc or r an t ( in mel ody ”mel ” ind = mellist [mel] [count]          ,   at the time ” count ” .
90
91         # Get lano te midi in the midi list = scale [ ind ]          ” scale ” .
92
93         # C al c ul of the frequency in Hertz .
94         freq = midiToHz ( noon )
95         # Chorus of 10 values around the captured frequency.
96         freqs = [ freq ÿ random . uni form ( . 9 9 1 . 0 1 ) for i inr an ge ( 1 0 ) ]
97         # Larger amplitude in 4 beats (emphasis on strong beats).
98         if ( count % 4 ) == 0:
99             f. mu =      . 1
100        else :
101            f. mu =      . 0 3
```

```
        # Increment the count (to advance to the next function call).
        count += 1
        # A c ti on salafin of the measure:
        if count == 1 6:
            # Reset count to 0 .
            count = 0
            # 50% of the time we get a new tune.
            if random . ran di nt ( 0 1 ) == 0:
                mel=random. ran di nt ( 0 3 )      ,              "
                print ( "new elisted' index: #A ssign              ,  email )
        frequencies to the oscillators and start the envelope.
        at . freq = freqs
        f. play ( )

# Call " new note 117 pat =      8 times per second wave.
P a t te rn ( time =.125                ,  func ti on=new note ) . play ( )

sec. gui( locals ())
```

scripts/chapter 07/01 text.py file

7.2 Revision

Grain generator with recursive delays. This script uses all the aspects developed so far. than here. Study each part carefully!

```
from pyo import *
import random

s = Server ( ) . boot()

############################################ ######################
### SNDS PATH is a string coming from the pyo module which points to ###
### folder where areinstalledless on sprovided with pyo . Don't use ###
### I have been constantly reading your own s on s. ###
########################################################## ############################
sound=SNDS PATH+ "/transparent. aif"

# Keep in memory the duration of the sound ( 2 nd element of the list that returns n info ) .
s on du r = sn di nfo ( sound ) [ 1 ]
print ( "Duration of sound: sound of r)

# L ist of midi notes to restrict transpo sition settings .
mnotes = [ 4 8 5 0 6 0 6 9 # C onve r if on in transposition factor at the end of the note 6 0,      5 3,  5 5,  5 7,      6 2,  6 5,  6 7,      ,  7 2]

transpose = [ midiToTranspo ( x ) for x in mnotes ]

####### Audi o p rocess #######

# Enveloping and setting up a sound player .
  f = Fader ( fad ei n =.005 fadeout =.01 hard =.02) ,
sf = S fPlayer ( sound , speed=1, mul=f )

```

```
# Bank of delays (4 delays with different feedbacks).
dls = Delay ( sfd el ay = [ .1 , .2 , .3 , .4 ] feedback = [ .2 5 , .5 , .3 5 , .7 5 ] maxdelay=1,          ,
                         mul=.25)
# D isposition in the stereo space of each of the relays.
pan = Pan( dl s , pan = [ 0 , .3 3 , .6 6 ] ,, sp re ad =.25)
# Reverb gener al e 34 re    ,   mix (2) for stereo reverb.
v = WGVerb(pan.mix (2), cutoff=4000, b al=.15). out (0)  feedback=.8     ,

####### S ec ti oncontrol #######

# Management of inspection methods . See the "new stream" function for
# the meaning of these variables . Possible values 40 offsetmeth = 0        ,  0 or 1 .

speed meth = 0

# P o si ti on of the grain in the sound in seconds ( for method 1 aoffsetmeth ) .
start = 0

# Inc remen t on the position in seconds. Applied to each new stream call.
startinc = 0. 0 1

# F unc ti on called periodically by the P a t te rn object (below).
def new stream ( ):
        # globalvariable then qu global start            we want to lamo di fi er .


        # sioffsetmeth is 0                  ,  we draw a reading point at random.
        # Else #     ,  we advance in the sound according to the increment. Onne
        not declare the global increment then squ if offsetmeth ==   we don't lemo di fi e not!
         0:
                sf. offset=random. uni form ( 0 else :       ,   sound of r ÿ.1)

                start += startinc
                if start >= (sound of r ÿ.1): # Render alafin sound start = 0         ,  we go back to the beginning.

                sf. offset = start

        # if speed meth is 0 we draw a transpose from the transpose list.
        # Otherwise we trap a slight vi a ti on around 1 .
         if speed meth == 0:
                sf. speed = random . ch oi ce ( transpose )
         else :
                sf. speed = random . uniform ( . 9 9        ,   1 . 0 1 )

        # N ew gr ain ree .
         f. hard = random . uniform ( . 0 2       ,   . 1 )

        # Display of new values.
         print ( "offset:%.2f,speed:%.2f hard:%.2f" %(sf.offset                  ,  sf. speed ,     f. hard ) )

        # We start playing the sound and the amplitude envelope.
         sf. play ( )
         f. play ( )
```

```
82 # V a ri a ti ons on the call frequency of the new stream function.
83 vtime = Choice (choice=[.2.8 freq=1./2.4) · 4 ,  ,  1 . 2 ,  2 . 4 ] ,
84
85 # P a t te rn calls in a way that the function given in argument
86 # (new stream ÿ s an spartheses ÿ) . We ÿ must ÿ call the method play ( )
87 # of the P a t te rn object to activate it .
88 pat = P a t te rn ( time=vtime  ,  func ti on=new stream ) . play ( )
89
90s . gui( locals ())
```

scripts/chapter 07/02 grains delay.py

Chapter 8

Dictionaries

Two new types of data are presented in this chapter: the tuple and the dictionary.

8.1 The Tuple

A tuple, like a list, is a sequence of values separated by commas. On the other hand, a tuple is delimited by parentheses and is immutable. Particularly useful for creating lists that must not be modified in any way, the tuple is also faster to access than the listing.

```
>>> tup = ( 2 1 , 22 , 23 , 24 , 25 , 26 )
>>> tup [ 0 ]
21
>>> tup [ 2:5 ]
( 2 3 2 4 , 25 ) ,
>>> tup2 = ( 2 1 >>> , 1. 3 3 , ' Hello , [ 1 , 2 , 3 ] )
tup2 [ 1 ]
1. 3 3 0 0 0 0 0 0 0 0 0 0 0 0 1
>>> tup2 [ 2:4,]
( ' b on j or r [ 1,>>> , 2 , 3 ] ),
tup2 [ 1 ] = ' Hello
Traceback (most recentcalllast):
File ”<st di n>” , , line 1 in <module>
TypeError: you please objectd oe s not supp ort item as signmen t
```

8.2 The dictionary

The dictionary is a table of data, defined between braces {}, on the basis of key pairs - value. The peers are separated by commas and the key is separated from its value by the symbol two points (:).

```
my dict = { 1: 1 0 0 , 2: 1 0 1 , 3:102}
```

All valid objects in Python can be used as a key in a dictionary (integer, decimal, string, list, tuple, object, function, etc.) Values in a dictionary are accessed by calling it and giving it the key in square brackets.

```
>>> my dict = { 'a' : 1 2 3 ,
                12 : [ 1 3 H ello , 4 , 5 , 6 ] ,
                ( 1 , 2): World! ' }

>>> print ( my dict [ 'a ' ] )
 123
>>> print ( my dict [ 1 2 ] )
 [ 1 6 ] 2 , 3 , , 5 ,
4 >>> print ( my dict [ ( 1 H ell , 2 ) ] )
 o world!
```

For keys that are simple strings, it is possible to specify the key-value pairs by giving keywords to the constructor method (this aspect will be clarified in the next course when we discuss the concept of class) of the dictionary class.

```
>>> my dict = dict ( Montreal =514, Quebec=418, She rb ro oke =819)
>>> print ( my dict )
{ ' Montreal ' 5 1 4: 4 ,1 8Quebec': , 'She rb ro oke': 819}
```

A dictionary is not a sequence, so the elements are not listed in order. which means that it is impossible to predict the order of arrival of the elements, in a for loop by example.

```
>>> forclein my dict . key s ( ):
 . . .          print ( key my ,dict [ key ] )
 . . .
She rb ro oke 819
Quebec 418
Montreal 514
```

The keys method, which returns a list containing all the keys in the dictionary, can easily be used to loop through all the items in the dictionary. If the order is large, just put the list back in ascending order with the sorted function.

```
>>> forcleinsorted(my dict.key s()):
 . . .          print ( key my ,dict [ key ] )
 . . .
Montreal 514
Quebec 418
She rb ro oke 819
```

8.2.1 Dictionary operations

Some operations on a dictionary:

— Addition of elements:

```
>>> dict = { 'Montreal' >>> dict [  : 5 1 4  ,  'Quebec': 418}
                  'She rb ro oke '] = 819
>>> print ( dict [          'She rb ro oke '])
819
```

— Deletion of elements:

```
>>> del dict [          'Quebec']
>>> dict
{ 'Montreal' : 5 1 4        ,  'She rb ro oke': 819}
```

— List keys or values:

```
>>> dict. key s ( )
 ['Montreal' 'She rb ro oke' ]
>>> dict. values ( )
 [ 5 1 4 8 1 9]
```

— List all elements:

```
>>> dict. time ( )
 [ ('Montreal' 5 1 4 ) ,            ('She rb ro oke'     ,  8 1 9 ) ]
```

— Test the presence of a key (with the in keyword):

```
>>>   'She rb ro oke'     indicated
True
```

— Copy a dictionary (create an independent copy):

```
>>> new di ct = dict . copy ( )
>>> new di ct [ ' Quebec ' ] = 418
>>> print ( new di ct )
{ 'Montreal': 5 1 4 'Quebec': 4 1 8 >>> print(dict)  'She rb ro oke': 819}

{ 'Montreal' : 5 1 4        ,  'She rb ro oke'    : 819}
```

Note that the simple copy() method only works with dictionaries relatively, i.e. dictionaries which do not contain any containers among their items. If, for example, a list is associated with any key in the dictionary, such as the copy is not recursive, the list will not be independent of the source dictionary list. This implies that a modification applied to the list will be reflected in both dictionaries, although these are independent. The list remains a single object in memory with two references pointing to this memory space.

```
>>> d1 = { ' [ 0 p1 = d1 >> d2 , 2 , 5 , 7 , 1 0 ] }
copy ( )
>>> print(”d1=”,d1)
d1 = { ' p1 5 >>> d2 [ ' p1 , >>> , 7 , 1 0] }
print ( my dict ”d2=”] d2 ) = 9
d2 = { ' [ 0 p1 9] }
>>> print ( ”d1,=” d1 = , d1 )
{ ' [ 0 p1 2 , 2 , 5 , 7 , 9] }
```

The solution to this problem consists in using the copy module, in which we will find the deepcopy function allowing recursive copying of complex structures. With this function, not only the main container is copied, but also all the containers that are found among the dictionary items.

```
>>> import copy
>>> d1 = { ' 1 0 ] p1 : [ 0 , 2 , 5 , 7 ,
>>> d2 = copy . deepcopy ( d1 )
>>> print d1 , ( ”d1 , d1 )
= { ' p1 5 >>> d2 [ = p1 0 >>> , 7 , 1 0] }
print ( ”d2 =” d2 2 [ [ 0 ] p1 2 ”
d1 = { ' [ 0 p1 2 , d2 )
: , , 5 , 7 , 9] }
, , d1 )
: , , 5 , 7 , 1 0] }
```

Tutorial on the different data structures.

8.2.2 Construction of a histogram using a dictionary

Assuming the following sentence: programming in python is really fun.

The construction of a histogram, representing the frequency of repetition of each of the letters, is very simple to do if we base ourselves on a dictionary.

```
>>> text= “programming in python is really fun”
>>> letters = {}
>>> forcintext:
... letters [ c ] = letters . get(c) , 0 ) + 1
...
>>> print (letters)
```

```
{' at  : 6  ,        : 6  , ,  and  : 4  ,        : 1  ,  I  : 2  , , 'h' : 1  ,
 'm' : 4  ,  ' I ' : 1  ,  , oh  : 3 : ,  g'n': 5, 'p': 2;1  ,  ,  s  : 2  ,
  r  : 3  ,  'u' : 1  ,  you  5  ,  v  ,  there  : 1}
```

First, we create an empty dictionary: letters. Then we go through the character string, element by element. For each character, we call the method get of the dictionary class. This method returns the value associated with the key specified in first parameter. If this key is not present, the method returns a default value defined in the second parameter, in this case, the value 0. Thus, for each character, we question the dictionary in order to know how many times this character has passed since the beginning of the loop, we add 1 to the answer and replace with this value the old value associated to this character.

If we want to get the result for each letter in alphabetical order, we just have to convert our dictionary into a list of tuples, with the items method, then passing the list into the sorted function, to put them in order:

```
>>> letterssorted = sorted ( letters . items ( ) )
>>> print (letterstried),  –  ,  ,  ,
 [('6), ('6), ('l'1), ('m'4), (;2),  and  , 4),('g  ,  , 1),('h '  , 1),('i2),  ,  2),
 ('5), , ,  , ,  , 5),('1),o(',  , 3),('p'y1),,  ('1)]  r  , 3),
   s  ,  you  ,  ( ' ( 'n',( 'u '  v  ,  ,
```

8.3 Managing events over time by sending triggers

8.3.1 What is a trigger?

A trigger, or trig, is an impulse, i.e. an audio signal in which a sample with a value of 1.0 is surrounded by samples of 0.0 values. The rationale for this type of signal is to provide pyo with a high temporal resolution event management system. A trigger being an audio signal, it is possible to create parallel processing with a sample-accurate synchronization. There are two types of objects in the triggers category, objects that generate trigs and those that react to them.

Objects generating trigs

— Beat algorithmically generates a sequence of trigs.

— Exchange sends a trig when an audio input signal changes value.

— Cloud sends trigs, without particular rhythm, according to a density parameter.

—Metro _ generates trigs synchronously.

— Select sends a trig when the input signal matches an integer given as an argument.

— Thresh sends a trig when an input signal crosses a threshold given in argument.

— Trig sends a single trig to the play method call.

Objects reacting to trigs

— Counter increments an integer count.

—TrigChoice _ picks a value from a list given as an argument.

— TrigEnv starts playing a table of samples.

—TrigExpseg _ starts playback of an exponential envelope.

—TrigFunc _ calls a python function.

—TrigLinseg _ starts playback of a linear envelope.

— TrigRand generates a pseudo-random value.

— TrigXnoise generates a value according to various pseudo-random algorithms.

— TrigXnoiseMidi generates a Midi note according to various pseudo-random algorithms.

Consult the triggers category of the pyo manual for a complete list of objects participating in trig management.

8.3.2 Sequence of events

At first, we will concentrate on the objects allowing to create sequences of events, i.e. Metro and Beat. Here, with a metronome, how to change periodically, the frequency of an oscillator:

```
1 from pyo import ÿ
2
3s = Server ( ) . boot()
4
5 # Do not forget to use the method. play ( ) for Metro and Beat objects
6 met = Metro (time =.125) . play ( )
7 t = TrigRand (met , min=400, max=1000)
8 a = SineLoop ( freq=t ,              feedback =0.05     , mul=.3) . out ( )
9
10 sec. gui( locals ())
```

scripts/chapter 08/01 metro rand.py

Same process but this time by picking the values in a predefined range:

```
1 from pyo import ÿ
2
3s = Server ( ) . boot()
4
5 notes = [ midiToHz ( x ) for x in [ 6 0 6 met =        ,6 2 ,6 4 ,6 5 ,6 7 ,6 9 ,7 1 ,7 2 ] ]
Metro ( time =.125) . play ( )
7 t = T ri gCh oice (met , ch oi ce=no te s 8 a =      , port=0.005)
SineLoop ( freq=t ,                feedback =0.07     , mul=.3) . out ( )
9
10 sec. gui( locals ())
```

scripts/chapter 08/02 metro choice.py

The TrigChoice object acts the same way as the Choice object. On the other hand, it expects a trig to pick a value instead of picking it according to a given frequency. The Argument port allows you to specify a ramp time between the current value and the captured value.

Using Samples Stored in a Table

In order to get the effect of a new note played, it is necessary to apply an envelope amplitude on the signal. The TrigEnv object allows to start reading the contents of a table samples upon receipt of a trig (we will elaborate in more detail on the PyoTableObject type shortly). We will use here some objects of the PyoTableObject class in order to put in memory different amplitude envelopes.

— LinTable Builds an envelope from line segments.

— ExpTable Builds an envelope from exponential segments.

— HannTable Generates a bell-shaped envelope (Hanning).

LinTable and ExpTable expect a list of tuples representing each of the envelope points.

```
# ADSR envelope ( attack ÿ decay ÿ sustain ÿ release )
env = LinTable ( [ ( 0 ( 1 0 0 (5 ),0 , .5 ) , (,51,0 0 , .5 ) , ( 8 1 9 1 ,0 ) ] ,  size =8192)
```

The first value of the tuple represents the location, in samples, where to locate the point and the second value is the amplitude of the point. Thus, in the previous example, for a table of 8192 samples, we have the following points:

— point 1: value 0 at sample 0 (first location).

— point 2: value 1 at sample 100 (end of attack).

— point 3: value 0.5 at sample 500 (end of decay).

— point 4: value 0.5 at sample 5000 (end of holding).

— point 5: value 0 at sample 8191 (Release until the end of the table).

Let's add the envelope in the previous script.

```
1 from pyo import ÿ
2
3s = Server ( ) . boot()
4
5 env = LinTable ( [ ( 0 ( 1 0 0 ,(5) 0 , .5 ) ,,(15) 0 0 , .5 ) , 6 notes =  ( 8 1 9 1 ,0 ) ] ,  size =8192)
[ midiToHz ( x ) for x in [ 6 0 6 2 6 4 6 5 6 7 6 9 7 1 ,  ,  ,  ,  ,  ,  ,72]]
7
8 met = Metro (time =.125) . play ( )
9 t = TrigChoice (met , ch oi ce=no te s 10 amp  , port=0.005)
= TrigEnv (met , ta bl e=env 11 a( fr=eq=t, SineLoop ,  hard=.125 , mul=.5)
                              feedback =0.07  , mul=amp) . out ( )
12
13 sec. gui( locals ())
```

scripts/chapter 08/03 metro env.py

You can easily test the effect of an exponential envelope by replacing LinTable by ExpTable. To obtain a bell-shaped envelope, it suffices simply to create a HannTable object with no parameters. In the following script, a second metronome, many slower, has been added to trigger a global envelope on note generation.

```
1 from pyo import ÿ
2
3s = Server ( ) . boot()
4
5 env = LinTable ( [ ( 0 0 ) , 6 gl obenv  ( 1 0 0 , 1 ) ,  ( 5 0 0 , . 5 ) ,  ( 5 0 0 0 , . 5 ) ,  ( 8 1 9 1 , 0 ) ] ,  size =8192)
= HannTable ( )
7
8 notes = [ midiToHz ( x ) for x in [ 6 0                    , 6 2 , 6 4 , 6 5 , 6 7 , 6 9 , 7 1 , 7 2 ] ]
9
10 globmet = Metro (time=5) . play ( )
11 globamp = TrigEnv ( globmet ,                table=globenv , hard=5)
12
13 met = Metro (time=.125) . play ( )
14 t = TrigCh oice (met , ch oi ce=no te s 15 amp =      , port=0.005)
TrigEnv (met , 16 a = SineLoop ( freq=t,  poly=wheat = approx ,  hard =.125  , mul=.5)
                                  feedback =0.07       , mul=ampÿglobamp ) . out ( )
17
18 sec. gui( locals ())
```

scripts/chapter 08/04 metro env glob.py

One of the main dilemmas in sound synthesis and processing is the management of polyphony, that is, the duplication of processes. A quick example to demonstrate the importance of polyphony is simply extending the envelope duration of the previous script to .25 second to get longer, overlapping notes. What about the sound result?

There are clicks in the sound as you restart an envelope that had not yet finished its course. To lengthen the notes and create an overlap, it is absolutely necessary to have more than one instance of the same process. This will ensure that an envelope is completed before to be restarted, thus avoiding a discontinuity in the amplitude trajectory.

Objects that generate sequences of trigs (Metro, Beat and Cloud) all have an argument poly allowing to define how many audio streams will be slaved to a generation of pulses.

```
>>> a = Metro ( time =.125 , p ol y =4)
>>> print(len(a))
 4
>>>
```

If our metronome handles 4 streams of polyphony, the resulting sequence of objects will handle necessarily at least 4 streams of polyphony each. We can therefore duplicate a process entire sound simply by specifying a number of polyphony voices to the object Metro. For an envelope duration of .25 seconds, we need only 2 voices of polyphony.

```
from pyo import ÿ

s = Server ( ) . boot()

env = LinTable ( [ ( 0 0 ) , gl obenv ( 1 0 0 , 1 ) , ( 5 0 0 , . 5 ) , ( 5 0 0 0 , . 5 ) , ( 8 1 9 1 , 0 ) ] , size =8192)
= HannTable ( )

notes = [ midiToHz ( x ) for x in [ 6 0 ,6 2 ,6 4 ,6 5 ,6 7 ,6 9 ,7 1 ,7 2 ] ]

globmet = Metro (time=5) . play ( )
globamp = TrigEnv ( globmet , table=globenv , hard=5)

met = Metro (time=.125, p ol y=2). play ( )
t = TriChoice (met , , port =0.005) ch oi ce=no te s
amp = TrigEnv (met , , mul=.5) ta bl e=env , hard =.25
a = SineLoop ( freq=t , feedback =0.07, mul=ampÿglobamp )out ( )

sec. gui( locals ())
```

scripts/chapter 08/05 metro stereo.py

Playing a sound file stored in a table

With the TrigEnv object, it is very easy to loop a sound at a frequency determined by the speed of a metronome. This time we are going to place a sound in a table with the SndTable object and activate playback with a sequence of trigs. The getDur method of the SndTable object allows to retrieve the length of the table in seconds.

```
from pyo import ÿ

s = Server ( ) . boot()

snd = SndTable (SNDS PATH+"/accord.aif" )
hard = snd . getDur()
print ( "Duration: hard ) ,

met = Metro ( time=dur ) . play ( )
out = TrigEnv (met , table=snd , hard=hard ). out ( )

sec. gui( locals ())
```

scripts/chapter 08/06 sndtable.py

We can change the pitch of the sound by modifying the playing time of the table:

```
snd = SndTable (SNDS PATH+ "/accord.aif" )
hard = snd . getDur()
print ( "Duration: hard),

met = Metro ( time=dur ÿ2 ) . play ( )
out = TrigEnv (met , hard=hard ÿ2 , table=snd ) . out ( ) ,
```

If the playing time of the table exceeds the speed of the metronome, we add voices of polyphony to avoid clicks!

```
snd = SndTable (SNDS PATH+ "/accord.aif" )
hard = snd . getDur()
print ( "Duration: hard),

met = Metro ( time=dur ÿ2 , p ol y =2) . play ( )
out = TrigEnv (met , table=snd hard=hard ÿ 2 . 5 ) . out ( )
```

8.4 Creating musical algorithms

Rhythm

In this section, we will explore some algorithmic objects available in the pyo library. Consider a script similar to the example on the previous page, i.e. a metronome which activates an amplitude envelope and picks up a new frequency for an oscillator.

```
freqs = [ midiToHz ( x ) for x in [ 6 0 env = LinTable , 6 2 , 6 4 , 6 5 , 6 7 , 6 9 , 7 1 , 7 2 ] ]
( [ ( 0 ( 3 0 0 ( 1 0 0 0 , 0 ) , , 1 ) , , . 5 ) , ( 3 0 0 0 , . 5 ) , ( 8 1 9 1 , 0 ) ] , size =8192)

met = Metro (time=.125). play ( )

amp = TrigEnv (met , ta bl e=env hard =.125, , mul=.5)
fr = TrigCh oice (met , ch oi ce=freqs )
osc= SineLoop (freq=frfeedback=.08, mul=amp) . out ( )
```

We will first use the Beat object which is designed to generate rhythmic sequences in an algorithmic way according to weights given in argument. The parameters w1, w2 and w3 respectively control the percentage of chance of being present high, middle and low beats. The parameter taps indicates the number of beats in the measure (1 beat = a duration given to the time parameter). For example, to get a measure in 4/4 containing only the strong beats (the quarter notes), we would write this:

```
met = Beat( time =.125 , taps=16, w1=100, w2=0, w3=0)
```

To add 50% of the split times (the eighth notes):

```
met = Beat( time =.125 , taps=16, w1=100, w2=50, w3=0)
```

And Finally, 30% of the downbeats (all remaining sixteenth notes):

```
met = Beat( time =.125 , taps=16, w1=100, w2=50, w3=30)
```

Let's replace the script's metronome with a Beat object:

```
freqs = [ midiToHz ( x ) for x in [ 6 0 env = LinTable ,6 2 ,6 4 ,6 5 ,6 7 ,6 9 ,7 1 ,7 2 ] ]
( [ ( 0 ( 3 0 0 ( 1 0 0 0 ,0 ) , ,1 ) , ,.5 ) , ( 3 0 0 0 , . 5 ) , ( 8 1 9 1 ,0 ) ] , size =8192)

met = Beat( time =.125 , taps=16, w1=90, w2=50, w3=30) . play ( )

amp = TrigEnv (met , fr = your wheat = approx , hard =.125 , mul=.5)
TrigChoice (met , ch oi ce=freqs )
osc= SineLoop (freq=frfeedback=.08, mul=amp) . out ( )
```

This script will generate a new rhythmic sequence each time it is run. A new feature should be noted here. The Beat object not only gives trigs as signals, it also returns other useful information in the form of audio streams. You can access the different streams using dictionary-like syntax, calling the desired info, in the format string, in square brackets. Here are the different possible calls:

— obj['dur']: Audio signal containing the duration of the current note, i.e. the beat before the next trig.

— obj['amp']: Audio signal containing the amplitude of the current note, according to the high, middle and low beats.

— obj['end']: Audio signal containing a single trig on the last beat of the measure. It can be used to ensure that tempo changes occur at the beginning of a measure.

We are going to insert in our script the signals of duration and amplitude so that our algorithm behaves a bit more consistently:

```
freqs = [ midiToHz ( x ) for x in [ 6 0 env = LinTable ,6 2 ,6 4 ,6 5 ,6 7 ,6 9 ,7 1 ,7 2 ] ]
( [ ( 0 ( 3 0 0 ( 1 0 0 0 ,0 ) , ,1 ) , ,.5 ) , ( 3 0 0 0 , . 5 ) , ( 8 1 9 1 ,0 ) ] , size =8192)

met = Beat( time =.125 , taps=16, w1=90, w2=50, w3=30) . play ( )

amp = TrigEnv (met, dur=met[ 'dur' ], fr = your wheat = approx TrigChoice (met, ch oi ce=freqs), mul=met [ 'amp ' ] )

osc = SineLoop (freq=frfeedback=.08, mul=ampÿ 0.5). out ( )
```

Melody

Now that we've got the accents and durations handled, let's leave the rhythm aspect aside. to improve the melodic aspect of our algorithm. A melody is rarely made up randomly picked notes, even if the picking is done within a scale. In good standing In general, a melody is made up of (often conjoined) upward and downward movements. the bottom which can be spread over several notes. There are also repetitions of certain patterns. This means that our TrigChoice object does not really fulfill its role. melodic, it is too risky! There are two objects in the library which are more suitable for

generate melodic cells, TrigXnoise and TrigXnoiseMidi. As we wish to generate notes on tempered scales, we will use TrigXnoiseMidi, which first generates its melody into Midi notes, then can, on request, automatically convert to Hertz. This object implements different types of controlled random distributions that can be used to generate melodic motifs of all kinds. Learn more about controlled random distributions (which is a bit out of the context of this course), here is an excellent site with diagrams representing the distribution curves of the most popular algorithms:

Gallery of distributions

The algorithm that interests us in the present context is number 12, a generator of looping melody segments. Replace the TrigChoice line with the following line:

```
fr = Tri gXn oi seMidi (met ,  dist=12, x1=1, x2=.3  ,  scale =1, mrange =(60 ,84 ) )
```

The dist argument indicates the distribution number to use, see the list of algorithms available in the manual. Arguments x1 and x2 switch roles depending on the algorithm selected. In this case, they represent the maximum possible value (depending on the range Midi) and the greatest possible distance between 2 note pitches. The scale argument indicates the desired format for the obtained values (0 = Midi, 1 = Hertz) and the argument mrange is a tuple indicating the Midi register in which the freelance must be restricted.

Our script now gives a more consistent melodic profile, however we have lost our range. All ratings between 60 and 84 are now legal. A technique rudimentary but effective way to overcome this problem consists in choosing the note of the scale closer to the drawn note. The Snap object performs this task:

```
fr = Tri gXn oi seMidi (met , frsnap =  dist=12, x1=1, x2=.3 scale=0, mrange=(60 ,84 ) )
Snap ( frch oi ce =[ 0 osc = Sine(freq=fr  , ,2 ,3 ,5 ,7 ,8 ] , scale =1)
                                  feedback =.08 , mul=amp ÿ . 5). out ( )
```

The Snap object works with Midi notes, so don't forget to reset the scale argument to 0, to send Midi notes. It is now the Snap object that will take care of the conversion to Hertz, passing the value 1 to its own scale argument. In the choice parameter, we will give a list representing the first octave of the desired scale.

Here is the script in full:

```
1 from pyo import ÿ
2
3s = Server ( ) . boot()
4
5 env = LinTable ( [ ( 0  ,0 ) ,  ( 3 0 0 , 1 ) ,  ( 1 0 0 0 , . 5 ) ,  ( 3 0 0 0 , . 5 ) ,  ( 8 1 9 1 , 0 ) ] ,  size =8192)
6
7 met = Beat( time =.125  ,  taps=16, w1=90, w2=50, w3=30) . play ( )
8
9 amp = TrigEnv (met ,  your wheat = approx ,  hard=put [ ' hard '  ] ,  mul=met [ 'amp ' ] )
10
11 fr = Tri gXn oi seMidi (met , dist =12, x1=1, x2 =.3 scale =0, mrange =(48 ,85 ) )
12 frsnap = Snap ( frch oi ce =[,0 1 1] , scale =1) , 2 , 3 , 5 , 7 , 8 ,
```

```
13
14 osc = SineLoop ( freq=frsnap ,                feedback =.08 , mul=amp ÿ . 5). out ( )
15
16 sec. gui( locals ())
```

scripts/chapter 08/07 simple algo.py

Harmony

To create a two-voice counterpoint, simply duplicate the generator in place and modify the parameters in order to play on interesting registers and rhythms...

```
 1 from pyo import ÿ
 2
 3s = Server ( ) . boot()
 4
 5 env = LinTable ( [ ( 0          , 0 ) ,  ( 3 0 0 , 1 ) ,  ( 1 0 0 0 , . 5 ) ,  ( 3 0 0 0 , . 5 ) ,  ( 8 1 9 1 , 0 ) ] ,  size =8192)
 6
 7 met = Beat( time =.125 8 amp =  ,  taps=16, w1=50, w2=70, w3=50) . play ( )
 TrigEnv (met , 9 fr = TrigXn oi      your wheat = approx ,  hard=met[ 'hard' ] mul=met[ 'amp' ])
  seMidi (met , 10 frsnap = Snap ( frch oi     dist =12, x1=1, x2 =.3 scale =0, mrange =(48 ,73 ) )
ce =[ 0 11 osc = SineLoop ( freq=frsnap ,          , 2  , 3 75 , 1] , scale =1)
                                                 feedback =.08 , mul=amp ÿ . 5). out ( )
12
13 met2 = Beat( time =.125 taps =16, w1=100, w2=40, w3=0) . play ( )
14 amp2 = TrigEnv ( met2 hard=met2[ 'hard' ] mul=met2[ 'amp' ])
15   2 = Tri gXn oi seMidi ( met2 scale =0, mrange =(36 ,61 ), x2 =.3 16
frsnap 2 = Snap ( fr 2 ch oi ce =[ 0 1 1] , scale =1)  , 2  , 3 75 , 8 ,
osc 2 = SineLoop ( freq=fr sn ap 2                  ,  feedback =.08 , mul=amp2 ÿ          . 5). out ( 1 )
18
19 sec. gui( locals ())
```

scripts/chapter 08/08 stereo algo.py

8.5 Example: Small algorithmic counterpoint with 2 voices

```
#! /usr/bin/env python
# enc odin g: utf ÿ8
from pyo import ÿ
import random

s = Server ( ) . boot()

TIME =  . 2 # Duration of a tap
# Mode dictionary for choosing snotes
SCLS = { ' [ 0 p1 7  :  [0 ,2 ,5 ,  ,9] ,  p2  :  ,3 ,5 ,7 ,10] ,  p3  :  [0 ,2 ,7 ,8 ,11 ] }

# Amplitude Envelope
env = LinTable ( [ ( 0 0 ) ,  ,  (300 ,1 ) ,  (1000 ,.7) ,  (6000 ,.7) ,  (8191 ,0 ) ] ,  size =8192)

#8 ran di samultiply to notes to create a chorus
cho = Randi( min=.99 3 ) for i ,  max=1.01  ,  freq=[random. uniform ( 1  ,  inrange ( 8 ) ] )

met = Beat( time=TIME, taps =16, w1=70, w2=70, w3=50) . play ( )
amp = TrigEnv (met , ta bl e=env hard=met[ 'hard' ] mul=met[ 'amp' ])
fr = Tri gXn oi seMidi (met , dist =12, x1=1, x2 =.33 scale =0, mrange =(48 ,85) )
frsnap = Snap ( frch oi ce=SCLS [ ”p1” ] scale =1)  ,
osc = LFO( freq=frsnap ÿcho sh a rp =.75 , type=3, mul=ampÿ 0 . 1 ) . mixing ( 2 )

met2 = Beat( time=TIME, taps=16, w1=100, w2=40, w3=20) . play ( )
amp2 = TrigEnv (met2 hard=met2[ 'hard' ], table=env, mul=met2[ 'amp' ])
fr 2 = Tri gXn oi seMidi ( met2 scale =0, mrange=(36 ,61 ), x2 =.5 27  ,
frsnap 2 = Snap ( fr 2 ch oi ce=SCLS [ ”p1” ] scale =1)  ,
osc 2 = LFO( freq=frsnap 2 ÿcho  ,  sh has rp =.75 , type=3, mul=amp2 ÿ 0 . 1). mixing ( 2 )

re v = WGVerb( os c+osc2  ,  feedback=.7  ,  cutoff=4000, bal=.15) . out ( )

def newbeat ( ): # Generate new beat sequences
    put . new ( )
    met2 . new ( )

def change ( ): # actions at the end of each bar
    print ( 'end of measurement' )
    if ( random . ran di nt ( 0 1 0 0,) < 2 5 ): # 25% chance of picking a new range
        range = random . ch oi ce ( [ p2 frsnap1 . ch oi  ,  p3 ' ] )
        ce = SCLS [ range ]
        frsnap 2 . ch oi ce = SCLS [ range ]
        print ( ”  N ew range: , range)
    if (random. ran di nt(0 print  , 1 0 0 ) < 2 0 ): # 20% chance of picking up a new rhythm
        ( ”Changes pace” )
        newbeat ( )

# Call the function changes at each end of measurement
ch = TrigFunc (met[ 'end' ], change)

sec. gui( locals ())
```

scripts/chapter 08/09 counterpoint.py

Chapter 9

Classes and methods

9.1 Introduction to the concept of class and objects

The concept of class is one of the most important elements of Object Oriented Programming (OOP). An object is an instance created from a class, that is, from the definition of a particular type of object. The definition of an object, introduced by the keyword class, consists of a grouping of all the attributes and methods related to the management of a given process. From this definition, it is possible to create as many independent objects as necessary for the proper functioning of the program.
This type of programming makes it possible to build complex programs by organizing a set of objects, each having their role to play, which interact with each other. This approach is beneficial in that the different objects can be built independently of each other without any risk of interference, thanks to the concept of encapsulation. Indeed, the internal functionalities of the object as well as the variables which it uses to carry out its task are locked inside the object. To gain access from the outside, it is necessary to use well-defined procedures, which minimizes the risk of conflict.

The following concepts are a little beyond the scope of this course, but it is good to know that they exist. It is possible to create a class from the definition of another class, i.e. to reuse pieces of code already written to create a new, more specific functionality. This is possible thanks to the concepts of inheritance and polymorphism.

— The inheritance mechanism allows to build a child class from a parent class. The child class thus inherits the properties of its ancestor, to which we can add other features.

— Polymorphism makes it possible to attribute different behaviors, when calling a function for example, to objects deriving from each other, or to the same object depending on a certain context.

9.2 Definition of a simple class and its attributes

A class is created with the statement class followed by the name you wish to give it (for convention, class names always start with an uppercase letter). A class being a compound statement, so it is necessary to end the header line with the symbol colons (:) and indent all the code that makes up the body of the class. In contrary when defining a function, the name of a class is not necessarily followed by a pair of brackets. In fact, these are only necessary when we want to designate a class parent to the class we are building, a subject that will not be covered in this course. here is a example of definition of a class that will be used to create notes in Midi format:

```
classNote :
        " Instance of a Midi note"
```

This class, although useless for the moment, already allows us to create Note objects. Because that no functionality is yet defined inside the class, the only relevant operation would be to call the content of the documentation variable created automatically by Python!

```
>>> n = Rating ( )
>>> print(n.doc)   __       __
 Instance of a Midi note
```

Two details to note:

— The creation of an object is done by calling the class followed by a pair of parentheses (we will see later that arguments can be given when creating an object). This call returns an object that can be assigned to a variable using the equal symbol (=).

— To access the attributes or methods of an object, use the syntax object.attribute or object.method(). The period specifies to the interpreter to look for such attribute or such a method belonging to such an object.

Let's take the Note class and give it some default attributes, for example the height and the velocity of a midi note.

```
classNote :
        " Instance of a Midi note"
       pi tch = 48
       velocity = 127
```

Therefore, any object created from this class will have two attributes, pitch and velocity, with respective values of 48 and 127. It will now be possible to consult or modify the attributes of each of the objects created, independently of each other.

```
>>> n1 = Rating ( )
>>> print(n1. pi t ch 48       ,   n1. velocity )
 127
>>> n2 = Rating ( )
>>> n1 . pi t ch n1„ velocity = 6 0 100           ,
>>> print(n1.pi t ch n1.velocity)
 60 100
>>> print ( n2 . pi t ch 48    ,   n2. velocity )
 127
```

We have just seen how to define our own class allowing to create objects. The
An object's functionality is usually defined by the methods of which the class is made.
In the next section, we will see the steps to create our own methods and how to use these methods to modify the behavior of an object.

9.3 Definition of a method

We define a method as we define a function, with two differences however:

— The definition of a method must always be encapsulated inside a class, i.e.
that is, it must be part of the block of instructions defining the class.

— The first parameter used by a method must always be an instance reference.
By convention, the reserved word self is used to designate the instance. This means that he must always be at least one parameter between the parentheses when defining
of a method.

```
classNote :
        ”  Instance of a Midi note”
       def myFunction ( self ):
              print ( '     I am a method of class Note ' )
              print (self)
```

```
>>> n = Rating ( )
>>> no. my function ( )
I am a method of the Note class
< builtin. Note instanceat 0 x 1 c 0 f 1 f 3 0 >
```

When calling the function myFunction, we see that self corresponds to an instance
of the Note class.

To note :

Although the myFunction method definition has an argument in parentheses,
the call is made without arguments. This is because when calling a method with the syntax object.method(), the object itself is automatically given as the first argument

to the method, hence the definition with self as argument. If the method requires arguments particular, they will be added in the definition following the argument self. The call will be from standard way, like for a function, always ignoring the first argument (self).

```
classNote :
        " Instance of a Midi note"
        def f onc ti onA vecA rgs ( self print , x , y ):
                ( self )
                print (x,y)
```

```
>>> n = Rating ( )
>>> no. f unc ti onA vecA rgs ( 1 0 < , 2 0 )
builtin . Note instanceat 0 x 1 c 0 f 1 f 3 0 >
 10 20
```

There is a method, which is automatically called on creation of an object, allowing to initialize instance variables, this is the constructor method.

9.4 The constructor method

The constructor method is a special method that is called automatically when creating an object. This method must be called init (two underscore characters , the word init, then two more underlined characters).

```
classNote :
        " Instance of a Midi note"
        def _ _ initiate _ _ ( self ):
                self . pi tch = 48
                self . velocity = 127

        def ge tNo te (self):
                "Return lano te Midi"
                return (self. pi t ch , self . velocity )
```

In the code above, we see that the attributes are now initialized inside of the constructor method. We also note that the name of these attributes is preceded of the word self and a period. This makes it possible to define instance variables, i.e. variables that belong to the created object and that will be accessible anywhere inside the the object. Thus, the getNote function can return pitch and velocity values since it has access to instance variables of the object. If several instances are created from the same class, instance variables are independent from one object to another. It is therefore possible to change the self.pitch variable without risk of interfering with the pitch of others' notes instances of the class.

Let us now give a functionality specific to our class. In addition to returning the note Midi as is, we are going to define two more functions. The first will perform the conversion

of the Midi pitch in cycles per second (Hertz) while the second will transpose the velocity in normalized amplitude between 0 and 1.

```
classNote :
        ” Instance of a Midi note”
        def init(self):
                self . pi tch = 48
                self . velocity = 127

        def ge tNo te (self):
                ”Return lano te Midi”
                return (self. pi t ch , self . velocity )

        def mtof(self):
                ”Returns the frequency in Hertz of lano te return 8. 1 7 5 7 9 ”
                8 ÿ pow( 1. 0 5 9 4 6 3 3 self . pi tch ) ,

        def vtoa(self):
                ”Returns the amplitude of the note between 0 and 1”
                return self. velocity / 1 2 7.
```

In case you want to specify the pitch and velocity values of the note at the moment of the creation of the object, it would suffice to give these values in parameters at creation and retrieve them in the constructor method by placing arguments following the instance variable.

```
>>> class Note:
 . . .          ” Instance of a Midi note”
 . . .          def (self, v el): pi tself . pi tch = ,
 . . .                  init
 . . .                  pit
 . . .                  self . velocity = v el
 . . .
>>> n = Rating ( 3 6 , 1 1 7 )
```

Here is a small example of using the Note class to generate a melody.

```
1 from pyo import ÿ
2 import random
3
4 class Note:
5         ”Container for no te Midi”
6         def init , pi tself _ p( i self = pit): ,
7
8                 self . velocity = v el
9
10        def ge tNo te (self):
11                ”Return lano te Midi”
12                return (self. pi t ch , self . velocity )
13
14        def mtof(self): ”
15                ”Returns the frequency in Hertz of the note
```

```
16                 return 8. 1 7 5 7 9 8 ÿ pow( 1. 0 5 9 4 6 3 3 ,   self . pi tch )
17
18             def vtoa(self):
19                 ”Returns the amplitude of the note between 0 and 1”
20                 return self. velocity / 1 2 7.
21
22 number notes = 200
23 noteslist = [ ]   –
24
25 for i       inrange (num notes):
26             # Random draw with a value between 48 and 84
27             pit=random. r and r an ge ( 4 8 v el   ,  8 5 ,  2)
28             = random . ran di nt ( 1 0 3 0 )     ,
29             notes list . append(Note(pi tv el))       ,
30
31 s = Server ( ). boot()
32
33 # Initialization of empty lists to keep references to audio objects
34 fades = [ ]
35 oscs = [ ]
36
37 ### Initializing snotes ###
38 # args: delay = delay before activating the object 39 # the object (the stop() method is called automatically), dur = activation time of

40  for i inrange (num notes):  –
41             # Get a Note object
42             note = noteslist [ i ]
43             # Departure time lano te
44             start = i ÿ . 1 2 5
45             # Amplitude envelope. L f = Fader       ' amplitude is recovered by the method v toa ( )
46             ( fad ei n =.01 fadeout =.99 , hard=1,
47                             mul=no te . v toa()). play (delay=start, hard=1)
48             # O scissor. The frequency is retrieved by the mtof ( ) method
49             osc = SineLoop (freq=no te.mtof(), mul=f). out   feedback=.05      ,
50                                 ( i %2, d el ay=start # Add objects to      , hard=1)
51             lists
52             bland. append(f)
53             oscs. append(osc)
54
55 sec. gui( locals ())
```

scripts/chapter 09/01 class note 1.py

This technique allows us to manage the generation of notes in pure Python, environment very powerful in terms of algorithms. In the following script, we will replace the loop that picks up random values for pitch and velocity by a loop a bit more refined. Instead of picking pitches from the most complete chance, we will create a random walk by adding a small value to the previous height, thus minimizing jumps in the melody. For the velocity, we will impose accents by systematically giving a high velocity to the first note of each group of 4. The following script replaces the note generation loop by our new algorithm.

```
1 from pyo import ÿ
2 import random
3
4 class Note:
5         ”Container for no te Midi”
6         def init _ pi tself _ p( self = el):      ,
7
8                 self . velocity = v el
9
10        def ge tNo te (self):
11                ”Return lano te Midi”
12                return (self. pi t ch           ,   self . velocity )
13
14        def mtof(self):                                                    ”
15                ”Returns the frequency in Hertz of lano te return 8. 1 7 5 7 9
16                8 ÿ pow( 1. 0 5 9 4 6 3 3 self . pi tch )          ,
17
18        def vtoa(self):
19                ”Returns the amplitude of the note between 0 and 1”
20                return self. velocity / 1 2 7.
21
22 number notes = 200
23 noteslist = [ ]   -
24
25 pit = 64 # Initialization of the lah at the start
26 for i       inrange (num notes):
27        # Random walk. Adds an even value between ÿ4 and 5 to the last pitch
28        pit=random. r and r an ge (ÿ4, 5 2 ) + pit   ,
29        # Imposes slimites
30        if pit < 4 8:
31                pit = 48
32        elif pit > 8 4:
33                pit = 84
34        if ( i % 4 ) == 0:
35                v el = 40 # Highlights
36        else :
37                v el = random . ran di nt ( 8       ,   2 5 ) # Weak beats
38        notes list . append ( Note ( pi t          ,   v el ) )
39
40s = Server ( ). boot()
41
42 fades     ,   oscs = [ ]     ,   [ ]
43 for ino     inr an ge ( num notes ):
44        te = noteslist [ i ]   -
45         start = i ÿ . 1 2 5
46         f = Fader ( fad ei n =.01       ,   fadeout=.99 , hard=1, mul=no te . vtoa()). pl ay ( d el ay=start                  ,
                hard=1)
47         osc = SineLoop (freq=no te.mtof(), hard=1)       feedback=.05      ,   mul=f ). out ( i %2, d el ay=start          ,

48         bland. append(f)
49         oscs. append(osc)
50
51 sec. gui( locals ())
```

scripts/chapter 09/02 class note 2.py

9.5 Sample class in an external file

The principle of creating a personal library consists in writing a module containing a collection of classes that we can import at will in our scripts. Here is an example of module containing a class generating sound particles.

```
1 from pyo import ÿ
2
3 class P article:
4         def    _ _ initiate _ _ ( self , input ,          time =.125 , q=20,
5                                   maxdur=.2     fringe =[ 3 5 0        , 1 0 0 0 0] , p ol y =12):
6             # Amplitude envelope
7             self . table = LinTable ( [ ( 0 #          , 0 ) , ( 1 0 0 , 1 ) , ( 5 0 0 , . 3 ) , ( 8 1 9 1   , 0 ) ] )
8             InputFader allows crossfades on its source
9             self . input ut = InputFader ( input )
10            # All variable parameters are converted to audio
11            self . time = If g (time)
12            self . q = If g (q)
13            self . maxdur = If g ( maxdur )
14
15            # Gen ra ti on of trigs
16            self . metro = Metro (self.time, p ol y=p ol y ) . play ( )
17            # Pick up frequency and duration values
18            self . freq = TrigRand ( self.meter min=f range[0] max=f range[1])      ,
19            self . dur = TrigRand ( self . metro min=.01 max= self . maxdur )
20            # Reading the amplitude envelope
21            self . amp = TrigEnv (self . metro # Filtered ,   your blue = self . your blue    ,   hard= self . hard )
22            particle
23            self . filter = Biquadx ( self.input ,                    freq= self . freq , q= self . q,
24                                      type=2, stages=2, mul= self . amp)
25
26        defout (self):
27            ”Sends sound to speakers and returns ' self object. filter. out ( )                    himself.      ”
28
29            return self
30
31        def getOut(self):
32                ”””Returns the audio object that generates the output signal of the                    ”””
33            class in order to be able to include it in a processing chain. return self. filtered
34
35
36        def set I nput (selffade tim e =.05):
37            self . input . set Input ( xfade tim e )
38
39        def setTime(selfx):       ,
40            self . time. value = x
41
42        def setQ ( selfx ):    ,
43            self . q. value = x
44
45        def setMaxDur(selfx):       ,
46            self . maxdur. value = x
```

scripts/chapter 09/malibrairie.py

Then, all you have to do is import malibrairie into a script to have access to the classes which are defined there. In a second file (located in the same folder as malibrairie.py in order to to avoid problems), we import and use the Particle class:

```
#! /usr/bin/env python
# enc odin g: utf ÿ8
from pyo import ÿ
from malibrairie import P article
import random

s = Server ( ). boot()

### D ifferentessourcesono res ###
a = N oi se ( . 3 )
a1 = SineLoop (freq=50, feedback=.2, mul=1)
a2 = FM( carrier =100, ratio =5.67 inde x =20, mul=.3)

### 5 examples of using the Particle class ###
which = 0
if which == 0:
        b = Particle ( a 18 elif  ,  time=.125  , q=2, maxdur=.5) . out ( )
which == 1:
        qlfo = If ne ( .1 , mul=5, add=6)
        b = Particle ( a , q=ql fo , maxdur=.25) . out ( )
elif which == 2 :
        tlfo = If ne ( . 1 5 mul=.1 add =.15) ,
        qlfo = If ne ( .1 , mul=5, add=6)
        b = P article ( a time=tlfo , q=ql fo , maxdur=.25) . out ( )
elif which == 3 :
        b = P article ( inpu t=ac = P  ,  time =.25 , q=10, maxdur=.5 time  ,  fringe =[ 3 5 0 1 0 0 0] ) . out ( )
        article ( a 28 elif which ==4:  =.125 , q=5, maxdur=.25 fringe =[ 2 0 0 0 , 1 0 0 0 0] ) . out ( )

        tlfo = If ne ( . 1 5 add =.15)  mul=.1  ,
        qlfo = If ne ( .1 , mul=5, add=6)
        b = P article ( a , maxdur=.25)  time=tlfo, q=ql fod =
        Delay (b.getOut().mix(2), feedback=.75)  d el ay =.5  ,
        c = Free rb ( db al =.4) . out ( )  size=8, damp=.9,

def change ( ):
        y=random. ran di nt ( 0 if y  , 2)
        == 0:
                b. set Input ( a  ,  2)
        elif y == 1:
                b. set Input ( a1  ,  2)
        elif y == 2:
                b. set Input ( a2  ,  2)

#"pat. play ( )" enables automatic source switching
pat = P a t te rn ( change ,  5 )

sec. gui( locals ())
```

scripts/chapter 09/03 class particle.py

Chapter 10

External Controllers

Manipulating audio processes via external controllers is very effective in producing natural contoured parameter variations. Two communication protocols will be explored in this chapter. We are first going to dwell on the Midi protocol which will allow the creation of simple and effective virtual synthesizers. Then, we will elaborate on the OSC (Open Sound Control) protocol which offers a more generalized data control than the Midi protocol.

10.1 Midi Setup

To use a Midi interface in a python script, you must first ensure that the server audio (it is the Server object which takes care of the Midi inputs/outputs) receives the data from the interface desired. Two functions are available to test the Midi input configuration:

— pm list devices() displays all the interfaces connected to the system.

— pm get default input() returns the default interface number.

```
>>> pmlistdevices()
MIDI of vices:
 0: IN  , name: IAC Driver Bus 1          , Interface: CoreMIDI
 1: IN  , name: K eysta ti on Port 1      , Interface: CoreMIDI
 2: OUT, name: IAC Driver Bus 1           , Interface: CoreMIDI
 3: OUT, name: K eysta ti on Port 1       , Interface: CoreMIDI

>>> pm getdefa-ul tinput() -
0
```

If the default Midi interface is not the desired interface, just give the number from the interface to the setMidiInputDevice() method of the Server object before calling the method boot(). With the configuration above, to use the Midi Keystation keyboard, you must specify interface number 1:

```
s = Server ( )
s. se tMi di I npu tD e vice ( 1 )
s. boot()
```

Quick Tip: If the setMidiInputDevice() method receives a larger number than the largest number returned by the pm list devices() function, all available Midi interfaces will be available in the program.

10.2 Midi Synthesizer

This script implements a class generating a synthesis sound on reception of Midi notes.

```
from pyo import ÿ

class Synth:
        set init (selftranspo=1):
                # Transposition factor
                self . transpo = If g ( transpo )
                # R ec oi the Midi notes and assign them to 10 p ol yphony v oices
                self . no te = No tein (pol y =10, scale =1, first =0, last =127)
                # Processing of the height and amplitude sv al ues of Midi snotes
                self . pit = self. no te [ pitch' ] ÿ self. transport,
                self . amp = MidiAdsr (self.no te[ ' ], velocity sustain=.7 attack=0.001 ,
                                       decay =.1 , release=2, mul=.1)
                # Chorus of 4 oscillators # Panoramic , mixes in mono to avoid
                alternation of 10 voices of polyphony
                self . osc 1 = SineLoop (self. pi t mul= self. amp) . mixing ( 1 ) feedback=0.12 ,
                self . osc 2 = SineLoop (self. pit ÿ feedback =0.12 mul= self. amp) . mixing ( 1 )
                7 self . osc 3 = SineLoop (self. pit ÿ feedback =0.12 mul= self. amp) . mixing ( 1 )
                4 self . osc 4 = SineLoop (self. pit ÿ 1. 0 0 2 mul= self. amp) . mixing ( 1 ) feedback=0.12 ,
                # Stereo mixing
                self . mix = (self .osc 1+ self .osc 2+ self .osc 3+ self .osc 4 ) . mixing ( 2 )

        defout (self):
                " Activates the sending of the signal to the outputs and returns the object to itself. self . mix. out ( )

                return self

        def sig(self):
                "Returns the object that produces the final signal of the audio channel.
                return self. mixing

# Audio / midi con fi gura ti on
idev = pm getdefa ul tinput ( )
s = Server ( )
sec. se tMi di I npu tD e vi ce ( idev ) # Change idev according to your configuration
sec. boot()

a1 , a2 = Synth ( ), Synth ( .5 ) # octavereeletoctave lower

# The Sum of s si gn aux of ssynthspasse in a reverberation
re v = WGVerb( a1 . sig ( )+a2 . sig ( ) , cutoff =3500, b al =.3) . out ( ) feedback=8
```

```
sec. gui( locals ())
```

scripts/chapter 10/01 midi synth.py

Two new objects are introduced here, Notein and MidiAdsr. The first receives the notes of the Midi keyboard and generates two separate audio streams per note, one for pitch (self.note['pitch']) and one for amplitude (self.note['velocity']). The second generates a classic ADSR envelope depending on the velocity of the note. Here is a slightly improved version of our program!

```
from pyo import ÿ
from random import uniform

class Synth:
    set init _ _ ( self , transport =1):
        self . transpo = If g ( transpo )
        self . no te = No tein (pol y =8, scale =1, first =0, last =127)
        self . pit = self. note [ ] ÿ self . transport , 'pitch'
        self. amp= MidiAdsr (self.no te[ ' ] velocity attack=0.01 decay=0.1 sustain=.7 ,
                             , , release=1, mul=.3)

        self . osc 1 = LFO( freq= self . pi tself . , sh a rp =0.25 , mul= self . amp). mixing ( 1 )
        osc 2 = LFO( freq= self . pit ÿ 0. 9 9 7 self . osc 3 , sh a rp =0.25 , mul= self . amp). mixing ( 1 )
        = LFO( freq= self . pit ÿ 1. 0 0 4 self . osc 4 = , sh a rp =0.25 mul= self . amp). mixing ( 1 )
        LFO( freq= self . pit ÿ 1. 0 0 9 , sh a rp =0.25 mul= self . amp). mixing ( 1 )

        # Mix stereo (osc 1 etosc 3 on the left, self. mix = osc 2 etosc 4 adroit )
        Mix ([ self. osc 1+ self. osc3 , self . osc 2+ self . osc 4 ] , voices =2)

        # Distortion with LFO surled riv e
        self . lfo = Si ne ( freq=uni form ( . 2 , . 4 ) , mul=0.45 self . , add=0.5)
        distortion = Di sto ( self . mix d ri ve= self . lfo , slope =0.95 , mul=.2)

    defout (self):
        self . distortion. out ( )
        return self

    def sig(self):
        return self. distortion


# Audio / midi con fi gura ti on
idev = pm getdefa ul tinput ( )
s = Server ( )
sec. se tMi di I npu tD e vice ( idev )
sec. boot()

# modulation wheel = vi brato amplitude
ctl = Mi di c tl ( 1 , minscale =0, maxscale =.2)
bend = Bendin ( brange =2, scale =1) # Pi tch bend
lf = If ne ( freq =5, mul=c tl , add=1) # Vib rato

a1 = Synth ( If ÿ bend )
comp = Compress ( a1 .sig() , re thresh=ÿ20, ratio=6)
v = WGVerb( comp , feedback =.8 , cutoff=3500, b al=.3) . out ( )

```

```
sec. gui( locals ())
```

scripts/chapter 10/02 midi synth 2.py

This program introduces two new objects from the Midi library, Midictl, to recover the values of a continuous controller, and Bendin, to retrieve values from the Pitch controller Bend.

10.3 Midi Controllable Audio Loop Generator

The following class plays one or more sound files in a loop and provides control over the transpositions via the Midi keyboard.

```
from pyo import ÿ

class SndSynth:
    def init(self, path, first=0, last=127):
        # Place the sound in memory
        self . table = SndTable (path)
        # scale=2 converts midi notes to transpose factors
        self . not ei n = No tein ( p ol y =3, scale =2, first= firstlast=last )
        self . transpo = self. not ei n [ pitch' ] ÿ self. table. ge tR a te ( )
        self . amp = Port( self.not ei n[ '' ], , mul=.5)velocity risetime=0.001

        # Osc ÿ> loop a table
        self . osc = Osc (self. ta bl efreq= self, tr an sp oself. mix = self. , mul= self . amp). mixing ( 1 )
        osc. mix(2)

    defout (self):
        self . mix. out ( )
        return self

    def sig(self):
        return self. mixing

idev = pm getdefa ul tinput ( )
s = Server ( )
sec. se tMi di I npu tD e vice ( 9 9 )
sec. boot()

# Cl a vi er Midi separated into regions by minor third
# at the t or r of snotes 6 0 6 3 7 2 li = [ ] 6 6 , 6 9 , , 75

for i inr an ge ( 1 7 ):
    wire in am e = ' snd %i_. aif ' % i
    central key = 57 + i ÿ 3
    li . append ( SndSynth ( wire in am e , first=central key ÿ1, last=central key +1) . out ( ) )

sec. gui( locals ())
```

scripts/chapter 10/03 midi synth snd.py

10.4 Dummy Sampler

This class allows the triggering and transposition of sound playbacks via the Midi keyboard.
The Notein object sends a trigger on receipt of a note with positive velocity via the audio stream trigon (self.notein["trigon"]). The latter is used to enable table reads.

```
from pyo import ÿ

class SndSynth:
    set init _ _ _ _ (self, path, first=0, last=127):
        self . table = SndTable (path)
        self . not ei n = No tein ( p ol y =10, scale =2, first= firstlast=last )
        # Duration of reading of the table = durationtable / factortranspo
        self . hard = self . table. getDur()/self. not ei n [ pi tch ' ]
        self . amp = Port( self. not ei n [ velocity ' ] risetime =0.001 , mul=.5)

        # No tein ["trigon"] sends a trig onreceipt of a noteon.
        self . snd = TrigEnv (self. not ei n[ "trigon" ], mul= self. table= self . your blue ,
                              hard= self . hard , amp). mixing ( 1 )
        self . mix=self. nd . mixing ( 2 )

    defout (self):
        self . mix. out ( )
        return self

    def sig(self):
        return self. mixing


# List of midi interfaces
pmlistdevices ( )

# Replace the value in idev with the idev = pm desired interface
getdefa ul tinput ( )

s = Server ( )
sec. se tMi di I npu tD e vice ( 9 9 )
sec. boot()

# Cl a vi er Midi separated into regions by octave between DOs
li = [ ]
for i inrange ( 1 7 ):
    wire in am e = ' snd %i . aif ' % i
    firstkey = i ÿ 12 + 24
    li . append ( SndSynth ( wire in am e , first=firstkey , last= firstkey +11) . out ( ) )

sec. gui( locals ())
```

scripts/chapter 10/04 midi synth snd 2.py

10.5 Simple granulator

Presented here is a sound file granulator which will serve as a pretext for parameter control via the OSC (Open Sound Control) protocol.

```
1 from pyo import ÿ
2
3s = Server ( ) . boot()
4
5 env = HannTable ()
6 table = SndTable ( ”Beethoven.aif” )
7 tables. view()
8
9 # example 0 ÿ> gearshift an stoucheralah au teu r
10 # example 1 ÿ> random play position
11 ex = 0
12 if ex == 0:
13 pos = Phasor ( freq=table . ge tRa te ( ) ÿ 0. 2 5 elif ex == 1: , mul=table . get Size ( ) )
14
15 pos = Randi( min=0, max= table . get S ize ( ) , freq =2)
16
17 pits = [ 1 . 1 . 0 0 2 ] ,
18 hard = N oi se ( mul=.002 , add =.1) # slight variations in the length of sgrains
19
20 gr ai n = G r anul ator ( ta bl e=ta bl e , # ta bl ec on ten an tlese samples
21 env=env , # in vel oppe of sgrains
22 pi tch=pi t , # pi tchglobal of sgrains
23 pos=pos , # reading position # of the ree of , in samples
24 hard=hard , sgrains in seconds
25 grains =32, ba # number of grains
26 sedu r =.1 , # of the reference ree for the grains
27 # (if hard == basedur , no transpo )
28 mul=.1) . out ( )
29
30 sec. gui( locals ())
```

scripts/chapter 10/05 granulator.py

10.6 Granulator control via OSC protocol

For the following example, we will use two python scripts. The first takes the granulator from the previous example and modifies the controls to receive data via the protocol CSOs. The second script controls the playback position as well as the duration of the grains of the granulator by propagating the data on the port number that the first script listens to. Data control, in an OSC send, are identified using an address system. The addresses, specified in string format, use Unix path syntax, with the forward slash (/) as separator. Example: '/data/freq', '/data/amp', etc.

```
1 from pyo import ÿ
2
3s = Server ( ) . boot()
4
5 table = SndTable ( ”Beethoven.aif” )
6 env = HannTable ()
7
8 # Listen to port 9000 at destinations 9 rec = OscReceive '/pos and '/rand'
(port=9000, address=[ ' /pos ' , '/rand' ])
10
11 # rec [ ' / pos ' ] ÿ> reading position
12 # rec [ ' / rand ' ] ÿ> random for the duration of sgrains
13 pos = Port( rec [ ' /pos ' ] risetime=.05 falltime =.05 , , mul=ta bl e . get Size ( ) )
14
15 pits = [ 1 . , 1 . 0 0 2 ]
16
17 # slight variations in grain length
18 hard = N oi se ( mul=Cli p ( rec[ '/rand' ], min=0, max=.09) , add=.1)
19
20 gr ai n = G r anul ator ( ta bl e=ta bl e , # ta bl ec on ten an tlese samples
21 env=env , # in vel oppe of sgrains
22 pi tch=pi t , # pi tchglobal of sgrains
23 pos=pos , # reading position # of the ree , in samples
24 hard=hard , of sgrains in seconds
25 grains =32, # number of grains
26 ba sedu r =.1 , # of the reference ree for lesgrains
27 mul=.1) . out ( )
28
29 sec. gui( locals ())
```

scripts/chapter 10/06 osc receive.py

```
1 from pyo import ÿ
2
3s = Server ( ) . boot()
4
5 # An lfo that controls the playback positionno rm ali seebetween 0 and 1
6 lfo = If ne ( . 1 , mul=.5 , add=.5)
7
8 # Manual control over grain transposition
9 rd = If g ( 0 )
10 rounds. ctrl ( title=” T ranspo si ti on per grain ” )
11
12 # Send controls on port 9000 to destinations '/pos'/rand' and
13 # host is the IP address of the target machine. 1 2 7 . 0 . 0 . 1 = local machine
14 send = OscSend ([lfo ' ) , Pow( rnd , 6 ) ] , port=9000, address=[ ' /pos ' , '/rand' ] , host='
1 2 7 . 0 . 0 . 1
15
16 sec. gui( locals ())
```

scripts/chapter 10/07 osc send.py

Chapter 11

Cecilia5 API - Module Writer

11.1 Introduction

11.1.1 What is a Cecilia module?

A Cecilia module is a python file (with the extension ".c5", specific to the Cecilia application) containing a Module class, inside which the audio processing chain is developed , and an Interface list, allowing the software to build all the graphical controls necessary for the proper functioning of the module. The file can then be loaded by the application in order to apply the processing chain to different signals, whether they come from sound files or from the microphone input. The processes used to manipulate the audio signal must be written with the dedicated signal processing module pyo.

11.1.2 Literature

Cecilia's API (Application Programming Interface) can be viewed within the application, via the Show API Documentation command, under the Help menu. It is divided into two parts: on the one hand, there is the documentation necessary for the development of the module itself, i.e. the class where the chain of commands is described. processing to be applied to the audio signal. Next, the various graphical interface elements (widgets) available are presented. This chapter follows the same structure.

11.2 BaseModule

Any Cecilia module must include a class called "Module", in which the sound processing chain will be developed. This class, to function correctly in the Cecilia environment, must inherit from the "BaseModule" class, defined in the source code of Cecilia. It is the "BaseModule" class which is in charge of creating, internally, the links between the graphic interface, the external communications (MIDI, OSC) and the audio process itself.

11.2.1 Initialization

A module must be declared like this:

```
class Module (BaseModule):
    def init _ _ ( self ):
        BaseModule. initiate _ _ ( self )
        # Here comes the processingch ain . . .
```

As the file will be executed inside the Cecilia environment, it is not necessary to import pyo or the BaseModule class, which is referred to as the parent class. These components being already imported by Cecilia, they will be available when the file is executed.

Here is the detail of each line of the previous excerpt:

```
class Module (BaseModule):
```

Class header row. The class name must be “Module”. She does inherit from the “BaseModule” class, defined in Cecilia's code.

```
    def _ _ initiate _ _ ( self ):
```

Method “constructor” of the class. No arguments need be defined (except, of course, on, of self).

```
        BaseModule. _ _ initiate _ _ ( self )
```

Initialization (call of the “constructor” method) of the parent class. This step is essential so that the links between the interface and the audio process are created.

```
        # Here comes the processingch ain . . .
```

When everything is correctly initialized, we can develop our processing chain from signal!

11.2.2 Class audio output

In order to route the audio signal from the process to the application, “self.out” must absolutely be the variable name of the object at the very end of the processing chain. Cecilia retrieves this variable to forward the audio signal to the Post-Processing section and ultimately, to the audio output. A typical example of declaring the “self.out” variable would look like this:

```
self . out = Interp ( self.snd , self . dsp, self . drywet, mul= self . approx )
```

The Interp object allows to interpolate between the original signal (“self.snd”) and the modified signal (“self.dsp”) according to a potentiometer of the interface (“self.drywet”). A general amplitude envelope (“self.env”), defined as a graph line, is applied to the final signal.

11.2.3 Module Documentation

Information relevant to a good understanding of the behavior of a module can be given in the doc string of the class “Module”. The Cecilia user will be able to display the module's documentation on screen via the Show module info command in the application's Help menu.

11.2.4 Public attributes of the BaseModule class

These variables, defined at the initialization of the “BaseModule” class, contain information on the current configuration of Cecilia and can be used at any time in the management of the audio process.

- self.sr: Sampling frequency defined within Cecilia's interface.
- self.nchnls: Number of audio channels defined in the interface.
- self.totalTime: Total duration of the performance, defined in the interface.
- self.number of voices: Number of polyphony voices coming from a cpoly (see section on interface definition).
- self.polyphony spread: List of transposition factors from a cpoly.
- self.polyphony scaling: Value used to weight the amplitude of the process according to the number of polyphony voices (cpoly).

11.2.5 Public methods of the BaseModule class

The following methods are defined inside the “BaseModule” class and can therefore be used in the composition of the processing chain of the module.

- self.addFilein(name): Creates and returns an SndTable object based on the name given to a cfilein.
- self.addSampler(name, pitch, amp): Creates a sampler/looper according to the name given to a csampler. The function returns the audio signal from the sampler.
- self.getSamplerDur(name): Returns the duration, in seconds, of the sound loaded into a sampler.
- self.duplicate(seq, num): Duplicate the elements of a list seq according to the number of iterations requested to the argument num. Returns the new list.
- self.setGlobalSeed(x): Assigns a fixed value to the root of the random seed, via the pyo server. This function makes it possible to obtain the same algorithmic sequences at each performance.

miss.

11.3 GUI elements

The Cecilia (.c5) file, in addition to the audio class, should specify the list of graphical controls needed for the module to work properly. This list must bear the name “Interface” and be made up of calls to functions specific to the Cecilia environment. Here is an example where a sound player, a graph line, two potentiometers and a polyphony control are declared:

```
Interface = [
        cs ample r ( name=”snd” ) ,
        cgraph ( name=”env”      , „ label=” Over all Amp” ,          func =[ ( 0 1,) , ( 1 1 ),] ,      col=” blue 1    ” ),
        cslider ( name=” freq    „   ,  label=” Filter Freq” , min=20, max=20000 col=” g reen   ,  init=1000,
                        rel=” log      ,  u ni t=”Hz”     ,  1 ” ),                                          ”
        cslider ( name=”q” , u ni   label=” Filter Q” , min=0.5 , max=25, init=1, rel=” log ” ),                  ,
                        t=”x”          ,  col=” green 2
        cpol y ( )
]
```

Here is the list of available functions and the description of the interface element that they allow to create.

11.3.1 cfile

The cfilein function creates a drop-down menu allowing the user to load a sound in a table (memory space) and to use it in the processing module. When the user chooses a sound on the disc, all the file is traversed and the sound files which located there will populate the drop-down menu, allowing quick access to the various sounds. A click on the menu opens the standard dialog for browsing the hierarchy on disk and a clicking on the arrow to the right opens the drop-down menu.

Several cfilein can be defined for the same module, as long as they have a name different. They will appear one after the other, according to the order given in the list, in the ”Input” panel of the interface. The cfilein is responsible for creating a table whose number of channels is consistent with the number of Cecilia's channels at the time the performance is launched, and this, regardless of the number of channels in the sound file.

To retrieve the table thus created in the ”Module” class, we use the addFi lein(name) method defined in the ”BaseModule” class. We give the name of the cfilein as an argument to the method.

Example

```
class Module (BaseModule):
        def init(self):   - -
                BaseModule. initiate      - -  ( self )
                self . table = self. add Fil ei n( ” self.out =  source    ” )
                Osc (self.ta bl e                          ,   freq= self . table. ge tR a te ( ) ,          mul= self . approx ÿ      . 2)

Interface = [                                    „
                cfilein(name=” source label=” S, or rce Audio” ),
                cgraph ( name=”env” func =[ ( 0 label=” Over all Amp” ,                    , 1 ) , ( 1 , 1 ) ] ,   col=” blue 1    ” ),
]
```

Declaration

```
cfilein(name=”filein”, label=”Audio”)
```

arguments

- name: {str} Character string allowing communication between the interface element and the processing module. The same string must be given to an addFilein method.
- label: {str} Descriptive string that appears just above the drop-down menu in the graphical interface.

11.3.2 csampler

The csampler function creates a drop-down menu allowing the user to load a sound in a loop playback module and use the signal from it as the audio source in the processing module. When the user chooses a sound on the disc, the whole file is browsed and the sound files therein will populate the drop-down menu, allowing quick access to different sounds. A click on the menu opens the standard dialog for browsing hierarchy on disk and clicking on the arrow to the right opens the drop-down menu.

Several csamplers can be defined for the same module, as long as they have a name different. They will appear one after the other, according to the order given in the list, in the ”Input” panel of the interface. The csampler takes care of creating a signal whose number of channels is consistent with the number of channels of Cecilia at the time the performance is launched, regardless of the number of channels in the sound file.

Using the small triangle in the toolbox (to the right of the drop-down menu), the user can open the playback control window. He will then have access to various parameters, in particular, the starting point and duration of the loop, transposition, amplitude, etc.

To create the player in a loop and retrieve the signal in the ”Module” class, we use the addSampler(name, pitch=1, amp=1) method defined in the ”BaseModule” class. We give the name of the desired csampler to the ”name” argument of the method. Optionally, it is possible to assign variables to the ”pitch” and ”amp” arguments, in order to control, respectively, the pitch and amplitude of the reading. E.g.:

Example

```
class Module (BaseModule):
    def __init__(self):
        BaseModule.__init__( self )
        self.snd = self.addSampler( ”snd” )
        self.out = Mix( self.snd, voices= self.nchnls, mul= self.approx )

Interface = [
    csampler( name=”snd” ),
    cgraph( name=”env”, label=” Over all Amplitude ”, func =[ ( 0 ,1 ) , ( 1 , 1 ) ] , col=” blue ” ),
]
```

Declaration

```
csampler(name=”sampler”, label=”Audio”)
```

arguments

- name: {str} Character string allowing communication between the interface element and the processing module. The same string must be given to an addSampler method.
- label: {str} Descriptive string that appears just above the drop-down menu in the graphical interface.

11.3.3 cpoly

The cpoly function allows a simplified management of polyphony in a module. She put set up two drop-down menus in the menus and switches section (lower section left). The first allows you to select the number of voices that will be played simultaneously while the second allows to configure the distribution of the transposition factors (phasing, chorus, detunes or one of the many chords provided).

The ”BaseModule” class generates three variables related to the values specified by the cpoly. They can be used to adjust the number of iterations of sound processes present. in the module. If the module uses a csampler as the audio source, the polyphony is automatically taken into account even when playing the sound, so there is nothing to add. The variables created automatically are:

- self.number of voices: {int} Number of selected polyphony voices.
- self.polyphony spread: {list} List of transposition factors defined according to the dis chosen contribution.
- self.polyphony scaling: {float} Amplitude factor used to adjust the signal gain depending on tion of the number of polyphony voices.

`

Note: You cannot declare more than one cpoly per module.

Example

```
class Module (BaseModule):
    def init(self):
        BaseModule. init #( self )
        Duplicates transpositions based on number of channels.
        tran sX ch nl s = self . duplicate ( self . p olyph ony sp re ad self nch nl s )
        # Slider frequency ÿ transpo factors. Has precisely the amplitude.
        self . saw = LFO( freq= self . freq ÿ tr an sXchnl s mul= self . p ol yphonysc ali ng )
        # Mix the signals according to the number of channels.
        self . out = Mix( self . saw mul= self . env ÿ 0 . 1 ). nchnl s
                                                    voices= self

Interface = [
    cgraph ( name="env" , label=" Over all Amplitude " labe , func =[ ( 0 , 1 ) , ( 1 , 1 ) ] , col=" blue 1 " ),
    cslider ( name=" freq cp , l="Base Freq" ,max=1000, init=150, rel=" log , col=" red 1 " ),
    ol y ( )
]
```

Declaration

```
cpoly(name="poly", label="Polyphony", min=1, max=10, init=1)
```

arguments

- name: {str} Character string associated with the cpoly. This argument is present for backward compatibility purposes but should not be used.
- label: {str} Descriptive character string that will be displayed in the graphical interface.
- min: {int} Number of minimum polyphony voices.
- max: {int} Number of maximum polyphony voices.
- init: {int} Number of initial polyphony voices.

11.3.4 cgraph

This function creates a graph line which represents the evolution of a variable on the total duration of the performance. The graph allows to elaborate complex automations on different parameters of the sound process implemented in the module. We get the value continues the line, in audio, using the variable "self.name", where "name" must be replaced by the value given to the name argument of the cgraph function concerned. This variable is created automatically by the BaseModule class. The func argument allows to specify the initial shape of the line by giving a list of points in (time - value) format. Time values must be specified between 0 and 1. They will then be adjusted to the total duration at launch performance.

If the value True is given to the argument table, the function drawn in the graph will be loaded into memory in a table (PyoTableObject) rather than being read continuously. The variable "self.name" will then contain a memory table and not an audio variable.

Example

```
class Module (BaseModule):
    def init _ _ ( self ):
        BaseModule. initiate _ _ ( self )
        self . wave = Osc (self.mytable, self.out freq= self . freq , mul=.2)
        = Mix( self.wave , voices= self . nchnl s , mul= self . approx )

Interface = [
    # Amplitude envelope (con ti nued variable)
    cgraph ( name="env" func =[(0 l="Over all Amp" , ,1 ) , ( 1 , 1 ) ] , col=" blue 1 " ),
    # Waveform (in memory in a table)
    cgraph ( name="mytable " labe, l="Waveform Shape" , min=ÿ1, max=1, table=True func =[ ( 0 ÿ1 ) , ( 1 ,
                , size =32768,0 ) , ( . 2 , 1 ) , (. 7 5 , , col=" red 1 " ),
    5 # Frequency trajectory ( variablec on tinue cgraph ( name=" , 0 ) ] scaleloga ri t hmic ) "
    freq func =[ ( 0 , labe l="Base Frequency 2 5 min=20, max=10000 col=" or anle l0 ), ,
                ,2 5 0 ) , ( 1 , 0 ) ] ,
]
```

Declaration

```
cgraph(name="graph", label="Envelope", min=0.0, max=1.0, rel="lin", table=False, size=8192,
       unit="x", curved=False, func=[(0, 0.), (.01, 1), (.99, 1), (1, 0.)], col="red")
```

arguments

- name: {str} Character string associated with the graph line. A variable self.name will be created by the BaseModule class.
- label: {str} Descriptive string that will be displayed in the chart's drop-down menu. • min: {float} Minimum value of the line on the Y axis. • max: {float} Maximum value of the line on the Y axis. • rel: {str} Type of scale used for the line resolution. "lin" = linear scale, "log" = scale

  logarithmic.
- table: {boolean} If this argument is True, a table will be created rather than reading the line continuously. Allows you to create envelopes that are not dependent on the duration of the performance (eg envelope of grains in a granulation module).
- size: {int} If the argument table is True, size indicates the size of the table in samples. • unit: {str} Description of the line unit. Not used. • curved: {boolean} If this argument is True, the initial line will be drawn using segments
  curves rather than linear. Same effect as double-clicking on the row.
- func: {list of tuples} List of pairs of values (X, Y) indicating the initial shape of the line. The values in X represent the time, normalized between 0 and 1 while the values in Y represent the amplitude of the points, according to the minimum and the maximum.
- col: {str} Color associated with the line. See the list of colors available at the end of the section.

11.3.5 cslider

The cslider function adds a slider to the panel below the interface graphic.
Each knob is automatically assigned a graph line with which to record/edit/replay automations for the parameter in question. We get the current value of the potentiometer, in audio, using the variable "self.name", where "name" must be replaced by the value given to the name argument of the relevant cslider function.
This variable is created automatically by the BaseModule class. The func argument allows to specify an initial automation to the potentiometer by giving a list of points in (time - value) format. Time values must be specified between 0 and 1. They will then be adjusted to the total duration at the start of the performance.

If the up argument is True, no continuous variable will be created and a discrete value will be given to a method call each time the mouse is released (after a change of value on the potentiometer). The method must be defined in the class "Module" and have the name "self.name up", where "name" must be replaced by the value given to the name_argument of the relevant cslider function.

For the following potentiometer definition:

```
cslider ( name="num" labe,l="# of G r ain s res=" int ,      , min=1, max=256, init=32, slide=0,
            up=True , u ni t=" grs " )
```

We should find, in the "Module" class, a method such as:

```
def num up ( selfv al ue ):
    gr = int ( value )
    # do whatever you want here . . .
```

At any time, you can retrieve the current value of the potentiometer, in data, using the get() method of the variable created for the potentiometer. For the cslider defined above, we would retrieve the number of grains like this:

```
nGrains = int (self.num.get())
```

Declaration

```
cslider(name="slider", label="Pitch", min=20.0, max=20000.0, init=1000.0, rel="lin", res="float",
        gliss=0.025, unit="x", up=False, func=None, midictl=None, half=False, col="red")
```

arguments

- name: {str} Character string associated with the potentiometer. A variable self.name will be created by the BaseModule class.
- label: {str} Descriptive string that will be displayed to the left of the knob.
- min: {float} Minimum value of the potentiometer.
- max: {float} Maximum value of the potentiometer.
- init: {float} Initial value of the potentiometer.
- rel: {str} Type of scale used for potentiometer resolution. "lin" = linear scale, "log" = logarithmic scale.
- res: {str} Numerical resolution of the potentiometer. "float" = decimal values, "int" = values integers only.
- gliss: {float} Duration of the audio portamento, in seconds, applied to the values produced by the poten tiometer.
- unit: {str} Description of the potentiometer unit. Displayed to the right of the numeric value.
- up: {boolean} If this argument has the value True, the potentiometer will provide a new value only on mouse release. A method is then called with the current value in argument.
- func: {list of tuples} List of pairs of values (X, Y) indicating the initial form of automation associated in the graph. The values in X represent time, normalized between 0 and 1 while the values in Y represents the amplitude of the points, according to the minimum and the maximum. If a list is given to this argument, the read mode of the potentiometer will be automatically activated.
- midictl: {int} Assigns, at initialization, a midi controller to the fader.
- half: {boolean} Tells the potentiometer to use half the graphics space of a poten standard tiometer. Usually grouped in pairs.
- col: {str} Color associated with the potentiometer. See the list of colors available at the end of the section.

11.3.6 crange

11.3.7 csplitter

11.3.8 ctoggle

11.3.9 cpopup

The cpopup function generates a drop-down menu, inserted in the panel to the left of the knobs, allowing the selection of items from a predefined list.

Two variables associated with the menu will be automatically created by the BaseModule class. Variable names will be constructed as follows:

- self.name + "index": An integer indicating the position of the selected item in the menu.
- self.name + "value": The character string of the chosen item.

Where "name" must be replaced by the value given to the name argument of the cpopup function concerned. If the value "foo" is given to name, the variables will be named self.foo index and self.foo value.

If the value "k" is given to the argument rate, a method bearing the name of the menu (argument name) must be defined in the class. The method, which will be called automatically by the selection of a new item in the menu, will receive two argument values, the index and the string of characters selected. Example method declaration:

```
def foo ( self #     ,   index    ,   value):
        inde x ÿ> int
        # value ÿ> str
```

Example

```
class Module (BaseModule):
        def init _ _        _ _ ( self ):
                BaseModule. initiate    _ _ ( self )
                self . n oi se = N oi se ( . 1 )
                self . filt = Biquad (self . n oi seself . out    ,   freq=1000, q=2, type= self . typeindex )    _
                = Mix( self . filt                          ,   voices= self . nchnl s      ,   mul= self . approx )

        def type ( selfv al ue ):index    ,
                self . filt . type =india x

Interface = [
        cgraph( name="env" func=[(0 , 0) , (1 , 1) , (1 , 1) ] , cpopup(name="type , label=" F ilter    , 1 ) , ( 1 , 1 ) ] ,  col=" blue 1  „ " ),
        Type" , value =["Lowpass" , "Highpass "         ini t="Bandpass" , col=" g reen 1       ,
                                                 , "Bandpass" , "Bandstop" ] , rat e="k" )
]
```

Declaration

```
cpopup(name=”popup”, label=”Chooser”, value=[”1”, ”2”, ”3”, ”4”], init=”1”, rate=”k”,
       col=”red”)
```

arguments

- name: {str} Character string associated with the drop-down menu. A method with this name must be created in the audio class.
- label: {str} Descriptive string that will be displayed to the left of the menu.
- value: {list of str} List of character strings specifying the values to insert in the menu scrolling.
- init: {str} Character strings indicating the initial value of the menu.
- rate: {str} Indicates whether the menu is dynamic (“k”) or active only when the performance is launched (“i”).
- col: {str} Color associated with the drop-down menu. See the list of colors available at the end of the section.

11.3.10 cbutton

11.3.11 cgen

11.3.12 List of colors

Five colors, in four tones, are available for the interface design. We assign the desired color, in string format, to the col argument of the creation functions GUI elements. Here is the list of possible colors:

red1	blue1	green1	purple1	orange1
red2	blue2	green2	purple2	orange2
red3	blue3	green3	purple3	orange3
red4	blue4	green4	purple4	orange4

11.4 Presets

11.5 Examples

11.5.1 Variable state filter

The first example illustrates the use of a sampler/looper as the audio source for a process. The sampler implements looped playback of a sound file with control over the playback parameters. The signal is then passed through a controlled variable state filter by the interface potentiometers.

```
                    import pyo since the file will be executed 1 # No one is inside of
2 # C ecilia in an environment where pyo is already important.
3
4 # Definition of the audi o class. The name ÿ must absolutely ÿ be ”Module” and must
```

```
# also inherit from the "BaseModule" class ( internal definieal of C ecilia ) .
# It is the "BaseModule" class which is in charge of initializing the variables
# necessary for communication between the interface and the audi o class.
class Module (BaseModule):
        # The class documentation string can be reconsidered in C ecilia
        # with the command C t rl+I (Cmd+I s or s OSX)
        """

        State V a ri a bl e F ilter .

        This module implements lowpass , bandpass and hi ghpassfiltersinparall el
        and all ow the usertointerpolate on an axislp ÿ> bp ÿ> hp .

        Sliders under the graph:

                ÿ Cutoff /Center Freq: Cutofffrequforlp and hp (centerfreqfor bp)
                ÿ Filter Q: Q factor ( inverse of bandwidth ) of the filter
                ÿ Type ( lpÿ>bpÿ>hp): Interpolatingfactor between filters
                ÿ Dry / Wet: Mix between the originalsignal and the filteredsignal

        Dropdown menus and toggles on the bottom left:

                ÿ # of V oi ces : Number of voices pl ayed sim ul taneou sl y ( polyphony ) ,
                                        onl y availableatinitializa tion time
                ÿ Polyphony Chords: Pitch interval between voices (chords),
                                                onl y availableatinitializa tion time

        Graph onl ypar ame te rs:

                ÿ Over all Amplitude: The amplitude cur ve ap pli ed on the totaldura ti on
        """                                             of the pe rfo rm ance

        # Method " def init builder ", has only "self" as an argument
                - -     - - ( self ):
                # Call of the method " ( self ) constructor " of the parentclass
                BaseModule. initiate - -

                # The "addSampler" method of the BaseModule class allows you to retrieve
                # lesignalp rovenan td 'un s ample r (loop playback with crossfade)
                # defined in the interface. The string gives the referenced method
                # in the name of " cs ample r self . , defined in the list of controls.
                snd = self. addSampler( "snd" )

                # F iltreaetatvariable ( interpolates between lp , bp and hp) . The filter is
                # apply to the ample signal r ( self . snd ) and use 3 variables
                # created automatically by the BaseModule class: self . freq, self. qet
                #self. kind . These variables were created according to the argument
                # "name" of stroisslidersdefined in the interface.
                self . dsp = SVF(self.snd self.q, self.type $elf . frequency ,
                                "
                # " self . out ÿ must absolutely ÿ be the variable name of the audio signal o
                # of the end of the processing chain. C eciliagetrecipeva riable
                # to follow the audi signal over the PostÿP roces section if nget
                # ultimately self . drywet is the associated variable "
                                                        graudioutput o
                # at the cslider (name="drywet " . . . ) andself . env" is the associated variable
```

```
59                  # at the cgraph (name="env " . . . ) defined in the interface.
60                   self . out = Interp (self.snd,self.dsp,self.drywet,mul=self.env)
61
62 # D efinition of the graphical interface .
63 # The name of the list ÿ must absolutely ÿ be 64 I nterface = [
65          # cs ample generates a loop drive from son files
66           cs ample r ( name="snd" ) ,
67          # cgraph create a line in legraph r
68           cgraph ( name="env" label=" Over all Amp" , #
69          cslidergenerate a potentiometer with its associated graph line
70           cslider(name=" freq label=" C utoff /Center Freq" , min=20, max=20000 " ) ,
71                       init=1000, rel=" log u ni t="Hz" ,col=" g reen 1 cslider ( name="q" ,
72           label=" F ilter Q" , min=0.5 , max=25, init =1, rel =" log " ),
73                       u ni t="x" col=" g reen 2 labe
74           cslider ( name=" type rel=" , l="Type ( lpÿ>bpÿ>hp) " , min=0, max=1, init =0.5 " ),
75                       lin , u ni t="x" col=" g reen 3 cslider
76           ( name="drywet " labe l="Dry / Wet" rel=" lin" min=0, max=1, init=1, col=" blue 1" ),
77                       ,
78          # cp ol y implements the two menus used to manage p ol yphony
79           cpol y ( )
80 ]
81
82 # Everything after the next marker in a . c5 has been
83 # written by the application and concerns the ve ga rde of spresets.
84
85 ###################################
86 ##### C eciliareservedsection #####
87 #### P resets saved from the app ####
88 ###################################
```

scripts/chapter 11/StateVarFilter.c5

11.5.2 Sound looper with feedback

The second example illustrates the use of a table to perform processes on a sound loaded into RAM. An oscillator whose output signal modulates the position of its own playback pointer is used to create a frequency modulation of a sound by himself. The signal then passes through a filter in order to soften the sonic result. Note the use drop-down menu (cpopup) to select the desired filter type.

```
1  class Module (BaseModule):
2
3          S el f ÿmodulated frequency sound looper .
4
5          Sliders under the graph:
6
7                  ÿ Transpo si ti on: Transpo si ti onof the in put sound in cent ts ,
8                  ÿ Feedback: Amount of self ÿmodulation in sound playback
9                  ÿ F ilter Frequency : Frequency , in Hertz , of the filter
10                 ÿ F ilter Q : Q of the filter ( inverse of the bandwidth )
11
12         Dropdown menus and toggles on the bottom left:
```

```
        ÿ Filter Type: Type of the filter
        ÿ # of V oi ces : Number of voices pl ayed sim ul taneou sl y ( polyphony ) ,
                                    onl y availableatinitializa tion time
        ÿ Polyphony Chords: Pitch interval between voices (chords),
                                          onl y availableatinitializa tion time

    Graph onl ypar ame te rs:

        ÿ Over all Amplitude: The amplitude cur ve ap pli ed on the totaldura ti on
                                          of the pe rfo rm ance

    """
    def   _ _ initiate _ _ ( self ):
         BaseModule. initiate     _ _ ( self )

        # C rea ti on of the table ( SndTable ) from the selected sound .
        # The add Fil ei ns method is responsible for ensuringthat the number of channels of
        # latableestc on se quen t with the number of channels of C ecilia .
         self . snd = self. add Fil ei n ( "snd" )

        # Convert from given values to transposition factor       " cents "
        # (CentsToTranspo). We multiply here by self . p ol yph on y sp re ad " # is the transposition list    Who
        given by the menu cp ol y # of the chosen chord and the number of p ol yphony voices # will contain  therefore,  active
        as many audio streams as # self list . trfactor = CentsToTranspo (self. tr an sp o        self . trfactor   "
                                                                     there are values in the
                    "  self . p ol yph on y spread "      .
                                                            ,  mul= self . p ol yph on y sp re ad )

        # The transposing factors are multiplied by the frequency at which
        # latabled oi tetrelue for obte ni rlah au teu roriginal sound .
         self . freq = Si g ( self . trfactor mul= self . snd . ge tRa te ( ) )

        # OscLoop works like SineLoop , # as an argument      but we give him lata bl ealire
        ( the sound file here ) . The feedback slider is multiplied
        # by a small value if not the test result is too hao ti c .
        # " self . p ol yphonysc ali ng is an amplitude value that adjusts for
        # decreases the volume as the number of voices increases (to keep a
        # constant amplitude).
         self . dsp = OscLoop (self.snd self.freq,self.feed ÿ0.0002 mul= self.p ol yphonysc ali   ,
                                   ng ÿ0.5)

        # We mix the si gn to the xs el on number of channels of C ecilia ( self . nch nl s ) .
         self . mix=self. dsp. mix(self.nch nl s)

        # Output variable (self . out) . A filter with slider control                "
        # (self.filtfself.filtq). is a created variable      self . filttindex
        # by the BaseModule class which contains the position of the chosen item
        # in the "filttself" menu. env" is the graph line for
        # the overall amplitude.
         self . out = Biquad ( self.mix          ,   freq= self . filtf , q= self . filtq,
                                   type= self . filttindex mul= self . approx )

    # The drop-down menu calls for each manipulation a method which carries
```

Machine Translated by Google


```
        # its name ("filtt" in this case). We absolutely have to define this
        # method with inde x and value arguments, to make the module work.
        def filtt(self index value):
            """
            index x ÿ> position of the item in the menu (integer)
            v al ue ÿ> v al eur selected in the menu ( string )
            """
            # We assign 'inde xal' argument self . out. type " type " of the Biquad
            =india x

Interface = [
        cfilein( name="snd" ),
        cgraph ( name="env" func,=[( 0, label=" T ransp o sition (u ni t=" , 1 ), , 1 ) ] , col=" blue 1 " ),
        cnts col=" red 1 " , ( 1 , min=ÿ4800, max=4800, init=0,
        , " ),
        cslider(name=" feed"" ), , label l="Feedback" , min=0, max=1, init=0.25 , u ni t="x" ,
                col=" pure pl e 1
        cslider ( name=" filtf " rel=" log , label=" F ilter Frequency u ni " , min=20, max=18000 col=" ,
                label=" F ilter Q" , init=10000 , t="Hz" , g reen 1 " ) ,
        cslider ( name=" filtq u ni t="x" , reen 2 " ), , max=25, init=1, rel=" log min=0.5 " ,
        ,
        # cpopup generates a drop down menu in the bottom left panel
        cpopup(name=" filtt label=" F ilter Type" , ini t="Lowpass" , col=" g reen 1 v al ue =["Lowpass" , "Highpass " ,
            " , "Bandpass" , B andreject " ] ),
        cpol y ( )
]
```

scripts/chapter 11/FeedLooper.c5

Chapter 12

Essential python modules

The sys and os modules offer a range of functions facilitating interaction with the operating system on which a script is executed. These functions allow the writing of a multi-platform code thanks to a simplified management of the operations specific to the different operating systems. In particular, we find tools for manipulating access paths (file paths) which take into account the differences in syntax from one system to another, particularly with regard to hierarchical delimiters (”/” under unix and ”\\” under Windows).

12.1 The sysmodule

Of all the features of the sys module, four will be discussed here, platform, maxint, path and argv.

12.1.1 sys.platform

The constant platform contains a string representing the operating system on which the script is executed. Major systems are identified by the strings ”darwin”
(OSX), ”win32” (Windows) and ”linux2” (linux distributions). This constant makes it possible to set conditions and apply operations specific to the different systems.

```
>>> import sys >>>
printsys . platform linux 2
```

12.1.2 sys.maxint

The numeric value associated with the constant maxint represents the largest integer that can be represented with the regular “integer” type. It allows to detect the architecture of the current python installation by comparing it with a power of two. If maxint is greater than 232/2, then the largest possible integer is defined with more than 32 bits, the architecture

is therefore 64-bit. We divide by two in order to take into account the fact that half of the integers defined with 32 bits are negative.

```
>>> import sys
>>> ifsys. maxint > 2ÿ ÿ3 2 / 2:
 . . .          a rch = 64
>>> else:
 . . .          a rch = 32
```

12.1.3 sys.path

The variable path contains a list of paths representing all the folders where python looks for its default libraries. The content of each of these files is accessible at all times to the interpreter. It is possible to modify this list during execution, either by adding or by deleting links, in order to have access to particular functionalities.

```
>>> import sys
>>> printsys. path                    ,                          ,                                     '
 [ ' /home/ olivier/ . epyo' /usr/lib/python2. 7 usr/lib/python2 . 7 /     ,    '/usr/lib/python2. 7 / pl a t ÿli nux 2 '/          ,
       lib ÿtk ' ' /usr/lib/python2 . 7 / li b ÿol d ,' li b ÿdynload /PIL ' . 7 / di st ÿp ack a ge      ,    '/usr/lib/python2. 7 /
       s / gtk ÿ2.0 . 7 / di st ÿp ack a ge s /ubuntu ÿone ÿclient gtk2 /usr/lib/python2 . 7 / di st ÿ
       chapter 0 2           ,    '/usr/lib/python2. 7 /di st ÿp ack a ge s /gst ÿ0.10 '/usr/lib/python2,
                                               ,    '/usr/lib/pymodules/python2. 7                  ,    '/usr/lib/python2
                                                            ,    '/usr/lib/python2. 7 / di st ÿp ack a ge s /wxÿ2.8ÿ
                              ,    ' /home/ olivier /Dropbox/ private / course /MUS2323/ coursenotes/        _    _
                                  _       ]
```

The following program illustrates the use of these features.

```
"""
The sys module:                    information specific to the operating system.

1ÿ Display operating system
2ÿ P ython installation architecture display
3ÿ Displayspa th s known to python
"""

import sys

pla atfo rms= {"darwin" : "OS X"             , "win32" : "Windows"          , " linux 2    " : "Linux"}
  if pla atfo rm s . has key_(sys.pl atfo rm):
        print "\nÿ Execute on platform %s \n" % pl atfo rm s [ sys . pla atfo rm ]
else :
        print "\nÿ Run on platform %s \n" % sys . pl atfo rm


#"/2" to account for half of the trails being negative
  if sys. maxint > 2ÿ ÿ3 2 / 2:
        print "ÿ The current python worm is 64ÿbit \n"
else :
```

```
22          print ”ÿ The current python worm is 32ÿbit \n”
23
24 print ”ÿ Python recognizes libraries 25 print sys .       ,   modules and scripts in these folders:\n”
path
```

scripts/chapter 12/01 essential sys.py

12.1.4 sys.argv

This variable is used to retrieve the arguments, in list format, given as an option when calling a python program from the command line. The name of the executed file is always at the first position of this list. If other options, separated by spaces, are given as arguments, they will be added to the list. The following script generates an additive synthesis whose oscillator frequencies must be specified as arguments at runtime. If sys.argv contains less than two items, an error message and usage description are displayed, then the program exits immediately.

```
1 #! /usr/bin /env python 2 # enc
odin g: utf ÿ8 3 from pyo import ÿ


4
5 args = sys . argv
6
7 iflen ( args ) < 2: print
8           arguments
9           frequencies of oscillators”print ”Enter the: number of print ”The\n" print
10          " Usage: python 02 sysargv . py freq 1 freq 2 freq 3 . . . \n” exit()
11
12
13
14 s = Server (duplex=0). boot()
15
16 print ”Running script %s ” % args [ 0 ]
17
18 d el args [ 0 ] 19
num = len ( args ) 20
freqs = [ float ( x ) for x in args ]
21
22 a = If ne ( freq=freqs , mul=0.5/num) . out ( )
23
24 sec. gui( locals ())
```

scripts/chapter 12/02 sys argv.py

Here is a concrete use of command line options. This program allows you to play one or more sound files, with the option of looping and the possibility of modifying the audio driver. Before executing the process, the program analyzes the options given on the command line and keeps only the sound files to play. Run the program with no arguments, or with the -h option to display the documentation.

```
1 #! /usr/bin/env python 2 # enc odin g: utf ÿ8
```

```
3 import sys , time
4 from pyo import ÿ
5
6 def use ( ):                                                    ”
7         print ”\nUsage: ” play
8                 [OPTIONS] file 1 [ file 2 [ file 3 [ . . . ] ] ] \ not"
9         printprint ”Options:
10        print ” ÿl: ;Ac tivates de . print ÿj: Uses jacken gi                    ”
11        neinsteadofportau di o . ” ÿh : Shows thish el p . \ not"
12        print
13
14  if “ÿh” in sys. argv:
15        use ( )
16        exit()
17
18 loop = F al se
19  if “ÿl” in sys. argv:
20        loop=True
21        pos = sys. argv. index x( ”ÿl” )
22        of the sys . argv [ pos ]
23
24 audio o= ”pa”
25  if “ÿj” in sys. argv:„
26        audi o= jack”
27        pos = sys. argv. index( ”ÿj” )
28        of the sys . argv [ pos ]
29
30  iflen (sys.argv) < 2:                                          ”
31        print ”\nNot enough arguments . use ( )
32
33        exit()
34
35 s = S erver (audi o=audi o). boot()
36  if not serve rB ooted ( ):
37        exit()
38 sec. start()
39  if not s. get Is Started():
40        exit()
41
42 for snd in sys . argv [ 1: ] try :                :
43
44              info = sn di info ( snd )
45              hard ,ch nl ,s format , sample=info[1], info[2] loop=loop, mul=.5).    , information [ 3 ]  , information [ 4 ]  , information [ 5 ]
46              sf = S fPl ayer ( snd print ,   out ( )
47              ”\nFilename print                                  „ :    ,  snd
48              ”Duration print ”Sampling                          : „ ,  hard
49              Rate print ”Number of                              : „ ,  sr
50              channels; printprint ”Sample Type if               „ ,  ch nl s
51              loop:       File Format                            : „ ,  format
52                                                                 :    , sample , ”\n”
53
54                    dump= raw input ( ”Hit Enter tostop pl a yi ng the sound.” )
55              else :
56                    time. sleep (hard +0.25)
```

```
57                  d el sf
58          except :
59                  print
60                  print snd con ,   "must be a WAVE or AIFF file! "
61                  ti nude
62 sec. stop ( )
63 times. sleep(.2)
```

scripts/chapter 12/05 commandline player.py

12.2 The os module

The os module contains all the tools needed to interact with the folder hierarchy on the hard disk. Here is a brief overview of the most common functions.

12.2.1 os.getcwd and os.chdir

The getcwd function returns the current working directory, i.e. the folder where the command line executing the python program was launched. Some publishers of text, E-Pyo is one of them, systematically modifies the current folder to reflect the folder where the executed file resides. The current folder is automatically added to the sys.path variable, which explains that files at the same hierarchical level as the program are accessible without providing the full path.

```
>>> import os
>>> prints . getcwd()
 /home/ olivier /Dropbox / private / course /MUS2323/ coursenotes / scripts / chapter 0 2
```

If, for some reason, usually to allow access to resources particular, the current folder must be modified, we will use the os.chdir function. This function accepts relative paths starting from the current folder (including ".." to go back one level) and full paths (i.e. from the root of the computer).

```
>>> bones. ch di r ( " . . / . . ")
>>> prints . getcwd()
 /home/ olivier /Dropbox / private / course /MUS2323/ coursenotes
```

12.2.2 os.listdir

This function returns the list of all the files contained in the directory given in argument. We will use the command os.listdir(os.getcwd()) to obtain the list of saved files in the current directory.

```
>>> import os
>>> prints . listdir(os.getcwd())
[ 01 sysargv . py 03 essentials bones . py 04 ospathessentials . p' ]
  c omm andline pl a ye r . py
```

12.2.3 os.mkdir, os.rmdir and os.remove

The os.mkdir function allows you to create a folder programmatically. without the path full access, the folder will be created from the current folder. If a folder with the same name already exists at the requested location, python will return an error. If a whole hierarchy folders should be created by the operation, os.mkdirs will take care of creating the missing folders.

The functions os.rmdir and os.remove can be used to, respectively, destroy a folder or file. Regarding the destruction of a folder, the latter must be empty, otherwise python will return an error. For more advanced file management functions, you will have to consult the shutil module.

```
>>> import os
>>> prints . getcwd()
/home/ olive tree
>>> bones. mkdir("temp")
>>> bones. ch di r ("temp")
>>> prints . getcwd()
/home/ olivier /temp
>>> bones. ch di r ( " "·)·
>>> bones. rmdir("temp")
```

Here is a small example of managing files on the hard disk:

```
#! /usr/bin/env python
# enc odin g: utf ÿ8
bone import

print "\nÿ D irector rec ou r an t: " , bones. getcwd()

print "\nÿ C rea ti ond 'a 'temp ' folder in the directory c or r an t " # mkdir returns an error iffolderexis
tedeja .
bones. mkdir( "temp" )

print "\nÿ Change of directoryc or r an t " os . chdir( 'temp' )


print "\nÿ C rea ti on of . . . inrange ( 4 ): "
for if =
        open ( "tm p fil eÿ%d.txt" % if.close() , "w" )


print "\nÿ Rec ou r an t directory: " , bones. getcwd()

print "\nÿ L ist of files held in the directory c or r an t: \ n"

 filelist=os. listdir(os.getcwd())
```

```
24 print file list
25                                                    ”
26 print ”\nÿ Delete if on files 27 for i inr an ge ( 4 ):

28             bones. remove( ”tm pfil eÿ%d.txt” %i)
29
30 print ”\nÿ Change directory or r an t (go back one level) ”
31 bones. ch di r ( ' · · ' )
32
33 prints
34 print ”ÿ Delete if on from folder 'temp ' ”
35 # Le dossier d oi tetre vi de!
36 bones. rmdir( 'temp' )
```

scripts/chapter 12/03 essential bones.py

12.3 The os.path module

The os.path module contains all the functions relating to the management of access paths. Whether for validity tests, the creation or modification of paths, you will nd all the necessary tools. Here is a brief overview of the common functions.

12.3.1 os.path.expanduser

This function is used to find out the path of the current user's folder. In order to ensure the portability of python programs, the ”ÿ” is considered as the user under all operating systems.

```
>>> import os
>>> bones. path. expąnder r ( ”˜” )
 ' /home/ olivier
```

12.3.2 os.path.isfile and os.path.isdir

Both of these functions are validity tests, whether a file or folder actually exists. well on the hard drive. Particularly useful, when launching an application, to check if the configuration files or the resource folder exist and run the code for them. created if they do not exist.

```
>>> import os
>>> prints . getcwd()
 /home/ olive tree
>>> prints . path. isdir(”Downloads”)
 True
>>> prints . path. isfile("weird.txt")
 False
```

12.3.3 os.path.join

This function is the ultimate tool for building paths that work on any computer. As the character which separates the hierarchical elements in a path is not the same on all systems (”/” under unix and ”\\” under Windows), the risk of The error increases if one tries to manage everything oneself. The os.path.join function works by concatenating path elements. Each argument given to the function is added to the character string and python takes care of inserting the correct delimiter according to the system.

So, to create the path to a sound in the download folder, we would write:

```
>>> import os >>>
path = os . path. join ( os . path . expanduse r ( ”~” ), >>> print path #     "Downloads"     ,     ”
                                                                                                    my daughter e. aif” )
on linux /home/ olivier /Downloads / my file e . aif
```

12.3.4 os.path.split and os.path.splitext

It is sometimes essential to know, during the execution of a program, where a file is saved, or what type of file it is. The split and splitext functions of the os.path module are then of great use. The role of os.path.split is to split a string (considered as a path) into two parts, the 'head' and the 'tail'.

The 'tail' is the last item in the hierarchical sequence and the 'head' is everything before it, thus giving the reference to the folder where the file is located.

```
>>> import os >>>
split = os . path. split(”/home/olivia/Downloads/ my fil e.aif”) >>> printsplit[0] /home/lisa/
Downloads >>> printsplit[1] my fil e. if
```

The os.path.splitext function, on the other hand, splits a character string in order to extract its extension. The separation is done based on the first point encountered while searching from the end of the string. Thus, to know the type of a file:

```
>>> import os >>>
split = os . path. splitext(”/home/olivier/Downloads/ my fil e.aif”) >>> printsplit[1]. aif >>>
split=os . path. splitext(”/home/oliver/notes.txt”) >>> printsplit[1]. txt
```

The following program creates a temporary folder in the current directory and saves a text file there. It reopens the file in read mode to retrieve the text and then cleans it up, i.e. the file is first deleted and then the folder is itself destroyed. Note the prior construction of the access paths using the os.path.join function!

```
#! /usr/bin/env python
# enc odin g: utf ÿ8
bone import

userpath = os . path. expander r ( "~" )

# C rea ti on of paths , by concatena ti on with os . path. join
temppath = os . path. join (u se rp a th , "temp" )
tempfile=os. path. join(temppath, " )
                                    template. txt
print "\nÿ User directory:
                                    ,   userpath

print "\nÿ Te s te lapresence d'un dossier '%s ' " % temppath
exist = os . path. isdir (temppath)

if not exist:
        print "\nÿ C rea ti on of folder '%s ' " % temppath
        bones. mkdir (temppath)

print "\nÿ Te s te lapresenced 'a file '%s ' " % tempfile
  exist = os . path. isfile (tempfile)

if not exist:
        print "\nÿ C rea ti on of file '%s ' " % tempfile
        f = open ( time p file "w" ) ,
        f. w ri te ( "ÿ This is the contents of tempfile .txtÿ" )
        f. close()

print "\nÿ Reading file '%s '\n" % tempfile

  f = open ( tem p file 32    ,    " r " )
print f . read ( )
f . close()

# bone. path. splitseparelaba se (folder path) from the file name
sppath = os . path. split(tempfile)
print "\nÿ File base path: , sppath [ 1 ],
                                    , sppath [ 0 ]
print "ÿ File name:

# bone . path. splitextseparate the file path of the extension if on
spext = bone . path. splitext(tempfile)
print "\nÿ Exten si on of the file:
                                    ,   span [ 1 ]

print "\nÿ Delete if on file '%s ' " % tempfile
bones. remove(tempfile)

print "\nÿ Delete if on folder '%s ' " % temppath
bones. rmdir (temppath)
```

scripts/chapter 12/04 os-path essentials.py

Chapter 13

Creating GUIs 1

13.1 Introduction to the wxPython graphics library

The Python programming language itself does not allow the creation of graphical interfaces. On the other hand, there are a certain number of modules giving access to libraries designed expressly for the development of man-machine interfaces. The two most common libraries today are pyQT and wxPython. They both offer a wide range of classes for implementing standard software behaviors. They also have the advantage of being multi-platform, that is to say that the same code will be executed correctly under the various most widespread operating systems (Windows, Mac OS and Linux). For this introduction to GUI design, we are going to use the wxPython library, which you must install independently on your system if you haven't already.

The resources

— The official wxPython website

— The wxPython download page

— Class by class documentation of wxPython

Caution

On the wxPython download page, you have the choice between the ”ansi” version and the ”unicode” version for the different versions of python. It is recommended to install the ”unicode” version because it allows the management of accents and other characters which are not part of the English grammar.

The 4 essential steps in building a GUI are:

— Management of the main execution loop of the software.

— The containers, ie the window and the panels where the interface will be exposed.

— The content, that is to say the objects offering an interaction with the user.

— The interaction between the manipulation of objects and the functionalities of the program, under the form of methods.

13.1.1 Execution loop management

The management of the main loop is done through a wx.App object or one of its derivatives. The wx.App object, which initializes the basic behaviors of the software, makes it possible to set up a wx application very quickly. An important point, which applies to any GUI library, is that the latter will eventually take ownership of the python execution loop, thus allowing the software to remain active for an indefinite period of time. ee. wx takes ownership of the loop when the MainLoop method is called on an object of class wx.App. This call is usually on the last line of the main script since subsequent lines will not be executed until the interface exits.

The following script starts by importing the wx library, then declares a wx.App object. An object of class wx.App must absolutely exist before starting to build interface objects. A window is then created (wx.Frame) and displayed on the screen (Show method). We will detail the containers in the following script. Finally, when everything is in place, we call the MainLoop method on our wx.App object. The execution loop then passes into the hands of our application and waits to receive events to execute various functions.

```
1 import wx
2
3 app = wx . App ( )
4
5 mainFrame = wx . Frame(None 6 , title=' Simple App ' )
mainFrame.Show()
7
8 app. MainLoop()
```

scripts/chapter 01/13 mainloop.py

When running this script a window like this should appear on the screen:

13.1.2 Containers

The program execution loop is now under interface control as long as this one does not leave. A container has already been created since a GUI cannot live without at least one window displayed on the screen. We will use 2 objects from the wx library as container. The first one (wx.Frame) is the main frame of our application, i.e. the window delimiting the space occupied by our application on the screen. In this window, we are going to create one or more panels (wx.Panel) where all the interface elements will be displayed of our program.

The objects provided in a graphic library generally constitute a base from from which we can build an interface with its own functionalities. The technique the most common is to create classes derived from a class in the library. We're going so define our own class (MyFrame) to display the application frame. Notice that the wx.Frame class is given in parentheses as a parent class to the MyFrame class. Our init method will receive the parameters needed to create the window and make them immediately follow the init method call of the wx.Frame class. For a class child does have all the characteristics of the parent class, the init method of the parent class must be called. Once the framework is properly installed, we will create a panel, with the wx.Panel object, which will be used to host the interface objects of our application. The SetBackgroundColour method allows us to specify a background color for our panel.

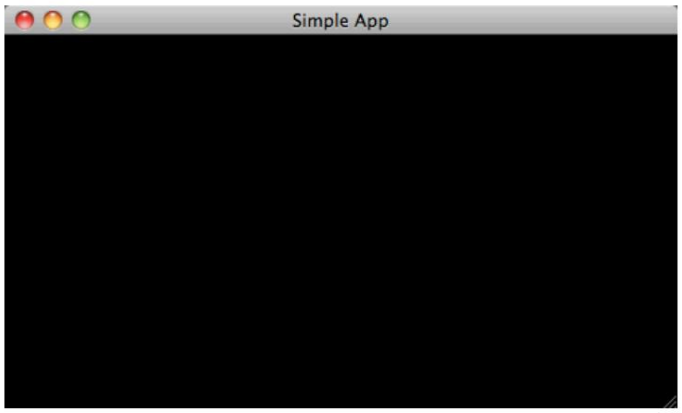
```
1 import wx
2
3 class MyFrame (wx.Frame):
4 def init(selfsize): _ , parent , , pos , title
5 init(self i d=ÿ1, title=Pa nel(self)self.p anel = wx . Frame . , pos=pos , size=size )
6
7 self . panel. SetBackgroundColour( ”#000000” )
8
9 app = wx . App ( )
10
11 mainFrame = MyFrame(None 12 mainFrame. Show() , title=' Simple App ' , pos = (100 ,100 ) , size = (500 ,300 ) )
13
14 app. MainLoop()
```

scripts/chapter 02/13 containers.py

Our application now looks like this:

Arguments when initializing the wx.Frame object

- self: The object itself goes directly through the method call. Never specify it during the execution.
- parent: The container in which our object is created. If it is a main window, which therefore has no container, we pass None. Note that when creating the panel (wx.Panel), self is indeed given as the first argument, telling the panel that it is created inside the window.



— id: Unique identifier of the object. -1 tells wx to automatically pick an identi
single filer.

• title: Character string displayed in the title bar of the window.

• pos: Tuple indicating the position (x, y) on the screen in pixels. (0, 0) is the top corner at
left.

• size: Tuple indicating the size (x, y) of the window in pixels.

13.1.3 Content

The content includes all graphical objects displayed inside the panels. We will use in this introduction the static text (wx.StaticText), the two-state button (wx.ToggleButton), the drop-down menu (wx.Choice) and the slider (wx.Slider). The initialization parameters will vary for different objects, except for the first two arguments, which will always be the container to put the object in and the unique identifier.

Let's add to our interface a two-state button that will allow us to start and stop our application's audio server.

Arguments when initializing the wx.StaticText object

• parent: The container in which our object is created.

— id: Unique identifier of the object.

• label: The displayed character string.

• pos: Tuple indicating the position (x, y) inside the container.

— size: Tuple indicating the size (x, y) of the object. wx.DefaultSize indicates to take a
default size according to the length of the text to display.

Arguments to the initialization of the wx.ToggleButton object

• parent: The container in which our object is created.

— id: Unique identifier of the object.

• label: The string displayed inside the button.

• pos: Tuple indicating the position (x, y) inside the container.

— size: Tuple indicating the size (x, y) of the object. wx.DefaultSize indicates to take a default size according to the length of the text to display.

```
1 #! /usr/bin/env python
2 # enc odin g: utf ÿ8
3
4 import wx
5
6 class MyFrame (wx.Frame):
7         def init_ parent, , pos=(self,wx.Frame.init i d=ÿ1, title=title,
8                 wx.Panel(self) _ _ _ _ parent, self.p anel =                    , pos=pos ,      size=size )
9
10                self . panel. SetBackgroundColour( ”#DDDDDD” )
11
12                self . onO f fText = wx . S ta ti c T ext(self.panel,=(28,10), self.onO    i d=ÿ1, labe l=”Audio” , pos
                  ff = wx.ToggleButton(self.panel,=(10,22), size=wx . D efa ul t Si ze )
13                                                                  i d=ÿ1, label l=”on / off ”          , pos
                                  size=wx . D efa ul t Si ze )
14
15 app = wx . App ( )
16
17 mainFrame = MyFrame(None 18 mainFrame. Show()    ,   title=' Simple App' , pos=(100,100),          size = (500 ,300 ) )
19
20 app. MainLoop()
```

scripts/chapter 13/03 togglebutton.py

Here is the state of our interface:

Now let's add a drop-down menu to select a sound from a predefined list.

Argument specific to the wx.Choice object

- choices: List of strings (which we can retrieve to change the source sound).

```
#! /usr/bin/env python
# enc odin g: utf ÿ8

import wx

class MyFrame (wx.Frame):
        def init_ parent, , pos (self) wx.Frame.init i d=ÿ1, title=title,
            wx.Panel(self) _ _     _ _ parent, self.p anel =                , pos=pos ,     size=size )

             self . panel. SetBackgroundColour( "#DDDDDD" )

             self . onO f fText = wx . S ta ti c T ext ( self . panel , i d=ÿ1, labe l="Audio" pos =(28 ,10) , size=wx . D ,
                                efa ul t Si ze )
             self . onOff = wx . ToggleButton ( self . panel , i d=ÿ1, labe l="on / off " pos=(10 ,28 ) , size=wx . D efa ,
                                ul t Si ze )

            # List of its content in the same folder as the script
             snd s = [ 'snd 1 . aif ' ' snd 2 . aif ' ' snd 3 . aif ' ' snd 4 . aif ' ' snd 5 . aif '  ,
                  ' snd 6 . aif' ]
             self . popupText = wx. St a ti c T ext ( self . panel , i d=ÿ1,
                   label=" C h ose a sound. . .                , pos = (10 ,60) , size=wx . D efa ul t Si ze )
             self . popup=wx. Choice (self.panel,       i d=ÿ1, pos =(8 ,78) ,
                             size=wx . D efa ul t Si ze     ,  choices=snds )

app = wx . App ( )

mainFrame = MyFrame(None    ,  title=' Simple App' , pos=(100,100),       size = (500 ,300 ) )
mainFrame.Show()

app. MainLoop()
```

scripts/chapter 04/13 popupmenu.py

The menu now appears in our interface:

Little trick!

The os module contains a number of utility functions. Two of these functions will be particularly useful in order to populate our menu with all the sounds present in the folder where our script is located at runtime.

— os.listdir(path) returns a list of all filenames in the given directory argument.

• os.getcwd() returns the current directory (usually the one from where the script is executed).

The following list generator therefore expresses:
For all files in the current directory, you add the file to the list only if its Last 4 characters are '.aif'

```
snd s = [ f for f in os . listdir(os.getcwd())                    if f [ ÿ4:] ==          '. aif' ]
```

Variant that includes in the menu sounds with the extensions '.wav' and '.aif'.

```
snd s = [ f for f in os . listdir(os.getcwd())                    if f [ ÿ4:] in [ '          . wave '  ,  '. aif'    ] ]
```

13.1.4 Interaction between interface and program functionality

The interaction between the manipulation of objects and the modifications to be made to the program is performed using a system of events to which methods are associated.

Events in wx are constants of the form wx.EVT EVENTTYPE. For example, the type of event sent when manipulating a toggle is wx.EVT TOGGLEBUTTON. – For the menu, it will be wx.EVT CHOICE.

We bind an event to a method using the Bind method of the wx.EvtHandler class (parent class of almost all wx objects).

```
obj. Bind ( event     ,   han dl er )
```

The called method (handler) will receive as argument an object derived from the class wx.Event containing information about the action that generated the event. In the example below, the handleAudio method, which will be called when manipulating the toggle, therefore takes 2 arguments: the object itself (self) as well as the generated event (evt).

```
def handleAudio(selfevt):      ,
    print event. GetInt()
```

Now let's attach a method to each of the objects in our interface. Handle them objects and watch the return in the console.

```
#! /usr/bin/env python
# enc odin g: utf ÿ8

import wx, os

class MyFrame (wx.Frame):
    def init, parent, , pos (self ), title, size ): wx.Frame.init i d=ÿ1, title=title,
        wx.Panel(self) parent, self.p anel = , pos=pos , size=size )

        self . panel. SetBackgroundColour( "#DDDDDD" )

        self . onO f fText = wx . S ta ti c T ext ( self . panel , i d=ÿ1, labe l="Audio" pos =(28 ,10) , size=wx . D ,
                            efa ul t Si ze )
        self . onOff = wx . ToggleButton ( self . panel , i d=ÿ1, labe l="on / off " pos=(10 ,28 ) , size=wx . D efa ,
                            ul t Si ze )

        # A toggle event calls the self method. handleAudio
        self . onOff . Bind (wx .EVT TOGGLEBUTTON, self .handleAudio )

        # bone . listdir ( path ) returns all rs files in the "path" folder
        # bone . getcwd ( ) returns the directory c or r an t .
        snd s = [ f for f in os . listdir(os.getcwd()) if f [ ÿ4:] in [ ' . wave ' , ' . aif' ] ]

        self . popupText = wx. St a ti c T ext ( self . panel , i d=ÿ1, „
                label=" C h ose a sound. . . , pos = (10 ,60) , size=wx . D efa ul t Si ze )
        self . popup=wx. Choice (self.panel, i d=ÿ1, pos =(8 ,78) ,
                size=wx . D efa ul t Si ze , choices=snds )

        # A menu event calls the self method. setSound
        self . popup . Bind(wx.EVT CHOICE, self.setSound)

    def handleAudio(selfevt): ,
        # event. Get Int() returns 1 if print evt. GetInt() letogglesta on , 0s ' ilestaoff


    def setSound(selfevt): ,
        # event. Get tS t ring ( ) returns the selected string in the menu
        print event. Get String ( )

app = wx . App ( )

mainFrame = MyFrame(None , title=' Simple App' , pos=(100,100), size = (500 ,300 ) )
mainFrame.Show()

app. MainLoop()
```

scripts/chapter 05/13 eventhandler.py

Our interface is now ready to manipulate an audio process. Initially time, we are going to write a small audio script that plays a table in a loop. We will write it in the global namespace so that our pyo objects are accessible from everywhere. Then there will remain change the methods called by events to control the audio process to Classes.

```
#! /usr/bin/env python
# enc odin g: utf ÿ8

import wx, os
from pyo import ÿ

s = Server ( ). boot()

# Loop sound player
table = SndTable ( "snd 1.aif" )
osc = Osc (ta bl efreq=ta bl e .ge tRa te ( ) )
mix = osc. mix(2). out ( )

class MyFrame (wx.Frame):
    def __init__(self, parent, title):
        wx.Frame.__init__(self, parent, id=ÿ1, title=title, pos=pos, size=size)
        self.panel = wx.Panel(self, id=ÿ1)

        self . panel. SetBackgroundColour( "#DDDDDD" )

        self . onO f fText = wx . S ta ti c T ext ( self . panel , i d=ÿ1, labe l="Audio" pos =(28 ,10) , size=wx . D efa ul t Si ze )
        self . onOff = wx . ToggleButton ( self . panel , i d=ÿ1, labe l="on / off " pos=(10 ,28) , size=wx . D efa ul t Si ze )
        self . onOff . Bind (wx .EVT TOGGLEBUTTON, self .handleAudio )

        snd s = [ f for f in os . listdir(os.getcwd()) if f [ÿ4:] in [self. popupText = wx. St a ti c T ext ( self . panel , i d=ÿ1,
                label=" C h ose a sound. . . ", pos =(10 ,60) , size=wx . D efa ul t Si ze )
        self . popup=wx. Choice (self.panel, i d=ÿ1, pos =(8 ,78) ,
                size=wx . D efa ul t Si ze , choices=snds )
        self . popup . Bind(wx.EVT CHOICE, self.setSound)

    def handleAudio(selfevt):
        if event . Get In t ( ) == 1:
            s. start()
        else :
            s. stop ( )

    def setSound(selfevt):
        table. sound = event. G e tS t ri ng ( )
        osc. freq = table. get tRa te ( )


app = wx . App ( )

mainFrame = MyFrame(None, title=' Simple App' , pos=(100,100), size = (500 ,300 ) )
mainFrame. Show()

app. MainLoop()
```

scripts/chapter 13/06 audio connections.py

A little more control

The 4 main steps for creating a GUI are now in place.
The last 2 will overlap each time a new control is inserted into the interface. A parameter that would be interesting to manipulate is the pitch of the sound. A potentiometer is ideal to accomplish this task. first we create a wx.Slider object to which we attach a new method. Note that the wx.Slider object only handles integer values.

```
# Explanatory text on the role of the poten ti om etre
 self . pi tT ext = wx . S ta ti c T ext ( self . panel , i d=ÿ1, labe l="Pi tch : 1. 0 pos = (140 ,60) ,
                                                    size=wx . D efa ul t Si ze )
# wx . Sliderfunc ti onalways fully
 self . pit=wx. Slider ( self . panel , i d=ÿ1, v al ue =1000, minValue =500,
                                          maxValue=2000, pos=(140,82), size=wx. D efa ul t Si ze )
# Attach the self method. ch an gePi tch at the event wx .EVT SLIDER
 self . pity . Bind (wx .EVT SLIDER, self .ch an gePi tch )
```

Arguments when initializing the wx.Slider object

- parent: The container in which our object is created.
- id: Unique identifier of the object.
- value: Value of the potentiometer at initialization (integer only).
- minValue: Minimum value of the potentiometer (integer only).
- maxValue: Maximum value of the potentiometer (integer only).
- pos: Tuple indicating the position in pixels in the container.
- size: Tuple indicating the size (x, y) of the object in pixels.

Next, we create the method self.changePitch which will take the return of evt.GetInt() (an integer representing the value of the potentiometer) in order to modify the “freq” attribute of our oscillator.

```
def ch an gePi tch (self #       ,   event):
        Manipulation of svalues to obtain a float between . 5 and 2 .
        x = event. Get Int ( ) ÿ 0. 0 0 1
        # S e tL ab el ( ) is used to modify the text displayed above the poten ti o meter
         self . pi tT ext. S e tL ab el ( "Pitch: %.3 f " % x )
        # We multiply our pitch factor by the frequency
        # alaquellelata bl eestlueasah au teu roriginale
         osc. freq = table. ge tRa te ( ) ÿ x
```

The setSound method must be adjusted to account for the value of the potentiometer when sound changes:

```
def setSound(selfevt):    ,
        # wx . Slider. GetValue ( ) returns the current value of the meter poten ti
```

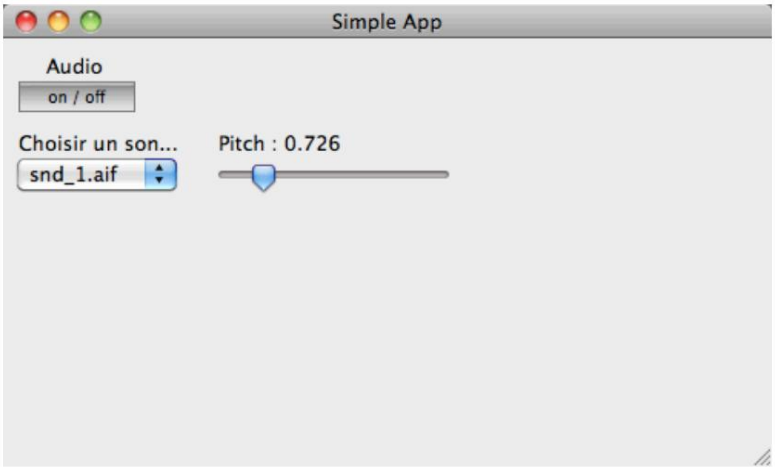
```
        x = self. pity . GetValue() ÿ 0. 0 0 1
        table. sound = event. G e tS t ri ng ( )
        osc. freq = table. ge tRa te ( ) ÿ x
```

We glue the pieces together and we get:

```
#! /usr/bin/env python
# enc odin g: utf ÿ8
import wx, os
from pyo import ÿ

s = Server ( ) . boot()
table = SndTable ( "snd 1.aif" )
osc = Osc ( ta bl efreq=ta bl e . ge tRa te ( ) )
mix = osc. mix(2). out ( )

class MyFrame (wx.Frame):
    def __init__(self, parent, pos, size, title):
        wx.Frame.__init__(self, parent, title=title, pos=pos, size=size)
        panel = wx.Panel(self)

        self.panel.SetBackgroundColour("#DDDDDD")
        self.onOffText = wx.StaticText(self.panel, id=ÿ1, label="Audio", pos=(28,10), size=wx.DefaultSize)
        self.onOff = wx.ToggleButton(self.panel, id=ÿ1, label="on/off", pos=(10,28), size=wx.DefaultSize)
        self.onOff.Bind(wx.EVT_TOGGLEBUTTON, self.handleAudio)

        snds = [f for f in os.listdir(os.getcwd()) if f[ÿ4:] in ['.wave', '.aif']]
        self.popup = wx.Choice(self.panel, id=ÿ1, pos=(8,78), label="Choose a sound...", pos=(10,60), size=wx.DefaultSize, choices=snds)
        self.popup.Bind(wx.EVT_CHOICE, self.setSound)
        self.pitText = wx.StaticText(self.panel, id=ÿ1, label="Pitch: 1.0", pos=(140,60), size=wx.DefaultSize)
        self.pit = wx.Slider(self.panel, id=ÿ1, value=1000, minValue=500,
                             maxValue=2000, pos=(140,82), size=(250,ÿ1))
        self.pity.Bind(wx.EVT_SLIDER, self.changePitch)

    def handleAudio(self,evt):
        if event.GetInt() == 1:
            s.start()
        else:
            s.stop()

    def setSound(self,evt):
        x = self.pity.GetValue() ÿ 0.0 0 1
        table.sound = event.GetString()
        osc.freq = table.getRate() ÿ x

    def changePitch(self, evt):
        x = event.GetInt() ÿ 0.0 0 1
        self.pitText.SetLabel("Pitch: %.3f" % x)
        osc.freq = table.getRate() ÿ x
```

```
app = wx . App ( )
mainFrame = MyFrame(None , title=' Simple App' , pos=(100,100), size = (500 ,300 ) )
mainFrame. Show()
app. MainLoop()
```

scripts/chapter 13/07 pitch slider.py



13.1.5 Two small improvements

Creating a class for the audio process

At the start of the script, instead of defining our audio process in the global namespace, we are going to create a small class in order to control the process from an object. The server will be left outside the class, which will leave the possibility of creating several instances of the Audio class while having only one server. Then, towards the end of the script, before creating the interface, we create an object from our Audio class and give it as an argument to the interface in order to be able to interact between the two.

```
#! /usr/bin/env python
# enc odin g: utf ÿ8

import wx, os
from pyo import ÿ

# We leave the external server then # cannot create more than we
one server per script
server = Server (). boot()

classAudio :
    def __initiate__ ( self ):
        self . table = SndTable ( "snd 1.aif" ) _
        self . osc = Osc (self. ta bl efreq= self. table. ge tRa te ( ) )
        self . mix=self. osc. mix(2). out ( )
```

```
17 class MyFrame (wx.Frame):
18         # New argument " audi o def init ( self
19          audi o ):_          _ _          , parent , , pos , title wx . Frame . init    size    ,
20                 ( self i d=ÿ1, title=title , parent , self . audi o = audi o                                    , pos=pos ,        size=size )
21
22                  self . p anel = wx . Panel ( self )
23                  self . panel. SetBackgroundColour( "#DDDDDD" )
24
25                  self . onO f fText = wx . S ta ti c T ext ( self . panel , i d=ÿ1, labe l="Audio" pos = (28 ,10) , size=wx .    ,
26                                                          D efa ul t Si ze )
27                  self . onOff = wx . ToggleButton ( self . panel , i d=ÿ1, labe l="on / off " pos=(10 ,28) , size=wx . D efa    ,
28                                                         ul t Si ze )
29                  self . onOff . Bind (wx .EVT TOGGLEBUTTON, self .handleAudio )
30
31                  snd s = [ f for f in os . listdir(os.getcwd()) if f [ÿ4:] in [self. popupText = wx. St a ti c T ext          ' . wave '  ,  ' . aif' ] ]
32                  ( self . panel , i d=ÿ1,
33                                  label=" C h ose a sound. . . , pos =(10 ",60) , size=wx . D efa ul t Si ze )
34                  self . popup=wx. Choice (self.panel,                    i d=ÿ1, pos =(8 ,78) ,
35                                                  size=wx . D efa ul t Si ze      ,   choices=snds )
36                  self . popup . Bind(wx.EVT CHOICE, self.setSound)
37
38                  self . pi tT ext = wx . S ta ti c T ext ( self . panel , i d=ÿ1, labe l="Pi tch : 1. 0 pos = (140 ,60) , size=wx . D  ",
39                                                         efa ul t Si ze )
40                  self . pit=wx. Slider ( self . panel , i d=ÿ1, v al ue =1000, minValue =500,
41                                                  maxValue=2000, pos = (140 ,82 ) , size =(250,ÿ1) )
42                  self . pity . Bind (wx .EVT SLIDER, self .ch an gePi tch )
43
44          def handleAudio(selfevt):          ,
45                  if event . Get In t ( ) == 1:
46                          server . start()
47                  else :
48                          server . stop ( )
49
50          def setSound(selfevt):       ,
51                  x = self. pity . GetValue() ÿ 0. 0 0 1
52                  self . audio o . table. sound = event. Ge tS t ri ng ( )
53                  self . audio o . osc. freq = self. audio o . your blue. ge tR a te ( ) ÿ x
54
55          def ch an gePi tch ( selfevt ):     ,
56                  x = event. Get Int ( ) ÿ 0. 0 0 1
57                  self . pi tT ext. S e tL ab el ( "Pitch: %.3 f " % x )
58                  self . audio o . osc. freq = self. audio o . your blue. ge tR a te ( ) ÿ x
59
60 app = wx . App ( )
61
62 # C rea ti onof an object of the "Audio" class
63 audio o = Audio ( )
64 # We pass our object as an argument to the Frame
65 mainFrame = MyFrame(None title=' Simple App' ,pos=(100,100),=audio)                    size = (500,300),          audi o
66 mainFrame . Show ( )
67
68 app. MainLoop()
```

scripts/chapter 13/08 class audio.py

The advantage of using an audio class rather than inserting the sound process at the global level of the script is that it becomes much easier to duplicate elements. Just simply create more than one Audio class object.

Duplicating elements to create stereophony

```
#! /usr/bin/env python
# enc odin g: utf ÿ8


import wx, os
from pyo import ÿ

server = Server ( ). boot()

class Audio:
        def  __initiate__ (self, out=0):
                # New argument , , to specify the output channel
                self . table = SndTable ( "snd 1.aif" )
                self . osc = Osc (self.ta bl e , freq= self . table. get tRa te ( ) ). or all of it )

class MyFrame (wx.Frame):
        # 2 arguments for audio objects o
        def init__, parent , __, pos=( self audioR ):Frame . init__, parent , i size , audioL ,
                variable for each audi o objec d=ÿ1, title=title # instance , pos=pos , size=size )

                self . audioL = audioL
                self . audioR = audioR


                self . p anel = wx . Panel ( self )
                self . panel. SetBackgroundColour( "#DDDDDD" )

                self . onO f fText = wx . S ta ti c T ext ( self . panel , pos =(28 ,10) , i d=ÿ1, label l="Audio" ,
                                                size=wx . D efa ul t Si ze )
                self . onOff = wx . ToggleButton ( self . panel , i d=ÿ1, labe l="on / off " pos=(10 ,28 ) , size=wx . D efa ,
                                                ul t Si ze )
                self . onOff . Bind (wx .EVT TOGGLEBUTTON, self .handleAudio )

                snd s = [ f for f in os . listdir(os.getcwd()) if f [ ÿ4:] in [ ' . wave ' , ' . aif' ] ]

                # Left channel interface
                self . popupTextL=wx. S ta tic T ext ( self . panel , i d=ÿ1,
                                label=" C h ose a sound. . . " , pos = (10 ,60) , i size=wx . D efa ul t Si ze )
                self . popupL=wx. Choice (self.panel, d=ÿ1, pos =(8 ,78) ,
                                                size=wx . D efa ul t Si ze , choices=snds )
                self . popupL. Bind(wx.EVT CHOICE, self.setSoundL)

                self . pi tTex tL = wx . S ta ti c T ext ( self . panel , i d=ÿ1, labe l="Pi tch : 1. 0 pos = (160 ,60) , size=wx . D ",
                                                efa ul t Si ze )
                self . pi tL = wx . Slider ( self . panel , i d=ÿ1, v al ue =1000, minValue =500,
                                                maxValue=2000, pos = (160 ,82 ) , size =(250,ÿ1) )
                self . pitL. Bind (wx .EVT SLIDER, self .changePitchL )

                # Right channel interface
```

```
48              self . popupTextR=wx. S ta ti c T ext ( self . panel , label=" C      i d=ÿ1,
49                      h o s e a sound . . . self . popupR = wx. Choice
                                                                  , pos = (10 ,160) , size=wx . D efa ul t Si ze )
50              ( self . panel ,                                  i d=ÿ1, pos =(8 ,178) ,
51                                                  size=wx . D efa ul t Si ze     ,   choices=snds )
52              self . popupR. Bind(wx.EVT CHOICE, self.setSoundR)
53
54              self . pitTextR=wx. S ta tic T ext ( self . panel , label="Pi      i d=ÿ1,
55                                      tch : 1. 0
                                                                  , pos = (160 ,160 ) , size=wx . D efa ul t Si ze )
56              self . pitR=wx. Slider(self.panel,                i d=ÿ1, value =1000, minValue =500,
57                                              maxValue=2000, pos = (160 ,182 ) , size =(250,ÿ1) )
58              self . pitR. Bind (wx .EVT SLIDER, self .changePitchR )
59
60      def handleAudio(selfevt):      ,
61              if event . Get In t ( ) == 1:
62                      server . start()
63              else :
64                      server . stop ( )
65
66      def setSoundL(selfevt):     ,                             "
67              "Changing its left channel. x = self. pitL. GetValue()
68              ÿ 0. 0 0 1
69              self . audioL. table. sound = event. G e tS t ring ( )
70              self . audioL. osc. freq = self. audioL. table. ge tRa te ( ) ÿ x
71
72      def changePitchL(selfevt):      ,                                                  "
73              "Playback speed change x = evt . Get Int ( ) ÿ     ,   left channel al .
74              0. 0 0 1
75              self . pi tTex tL . S e tL ab el ( "Pitch: %.3 f " % x )
76              self . audioL. osc. freq = self. audioL. table. ge tRa te ( ) ÿ x
77
78      def setSoundR(self "        ,   event):                   "
79              Sound change x = self.     ,   right al channel .
80              pitR. GetValue() ÿ 0. 0 0 1
81              self . audioR. table. sound = event. G e tS t ri ng ( )
82              self . audioR. osc. freq = self. audioR. table. ge tRa te ( ) ÿ x
83
84      def changePitchR(selfevt):      ,                                                  "
85              "Playback speed change x = evt . Get Int ( ) ÿ     ,   right al channel .
86              0. 0 0 1
87              self . pitTextR . S e tL ab el ( "Pitch: %.3 f " % x )
88              self . audioR. osc. freq = self. audioR. table. ge tRa te ( ) ÿ x
89
90 app = wx . App ( )
91
92 # 2 Audio objects 93      ,   output channel as argument
audioL = Audio()
94 audioR = Audio(1)
95
96 # We pass the 2 objects in the interface
97 mainFrame = MyFrame(None title=' Simple App' ,pos=(100,100),audioL=audioL              size = (450,280),
98                                  audioR=audioR)    ,
99 mainFrame . Show ( )
100
```

```
101 app. MainLoop()
```

scripts/chapter 09/13 stereo.py

13.2 Effects Selector

```
#! /usr/bin/env python
# enc odin g: utf ÿ8
import wx
from pyo import ÿ

6s = Server ( ) . boot()
sec. amp=0.5

class FxSwitch:
        """
        Audi o FxSwitch class. R eceives an audi i npu t and directs the signal
        to one of four available effects.

        Settings :

        input : PyoOb ject
                Sign al sourceamo di fi er . ,                ,
        fx: string { 'Delay' 'Chorus' Initial selection of the effect.      ,    'Reverb' } ,        optional
        """

        def    _ _ initiate _ _ (selffx=', input', ):#
                R eferences of svariablesreceived in argument
                self . input ut = input t
                self . fx = fx
                # C onve r si on of spar ame ters in audi o ( Si g ) with portamento (To)
                self . p1 = SigTo ( value =0.5 init =0.5,  time =0.05      ,
                self . p2 = SigTo ( value =0.5 init =0.5,  time=0.05       ,
                # The 4 effects available (Note the call to the stop() method)
                self . d el ay = Delay (self. input , feedback= self. p2 ) . stop (),
                self. distortion = Di sto ( self . input , drift= self p2 , sl ope= self .
                                                mul=.5) . stop ( )
                self . ch o rus = Chorus (self. input , depth= self. p1, mul=1.5) . stop      feedback= self . p2 ,
                                                ( )
                self . reverb = F reeve rb ( self.input , size= self.p1, damp= self.p2). stop ( )
```

```
        # Dictionary associating Audi objects with a string for future use
        self . fxdict = { 'Delay' : self. delay, :self. di sto 'Chorus' : self. chorus 'Reverb' : self. reverb }
        # Play the object given the self argument . fxdict [fx]. out ( )
                                                          "fx

    def changeFx ( selffx ):
        ' Method called by the menu # A rretel '   " C h ose an effect
        effectc or r an t
        self . fxdict [self. fx]. stop ( )
        # Start the new effect
        self . fxdict [fx]. out ( )
        # Replace the active referenceal ' effect
        self . fx = fx

    def changeP1 ( self """ A    ,   x):
        s sign a new value ( slider 1 ) to the SigTo which
            controllep remie r parameter of seffects. self . p1. value = x   """

    def changeP2 ( self """ A    ,   x):
        s sign a new value ( slider 2 ) to the SigTo which
            controls the second effect parameter. self . p2 . value = x   """

class MyFrame (wx.Frame):
    """
    Graphic interface .
    The parameter swi tcherat extends an object of the class def init ( self , pa ren    'FxSwitch'   .   """
    t=None titl e="Fx Swi tcher pos = (100 ,100 ) ,
                                size = (400,140) ,        s wi tcher=None ):
        # Call of the init method of the parentclass in
        # passing it the window's own arguments.
        wx . Frame . init i d=ÿ1, title=title, , pos=pos ,   size=size )
        # FxSwitch object reference
        self . s wi tcher = s wi tcher
        # C rea ti on of the panel
        self . p anel = wx . Panel ( self )
        self . panel. SetBackgroundColour( "#DDDDDD" )

        # Toggle to control server activation
        self . onO f fText = wx . S ta ti c T ext ( self . panel , pos =(28 ,10) ,   i d=ÿ1, label l="Audio"   ,
                                      size=wx . D efa ul t Si ze )
        self . onOff = wx . ToggleButton ( self . panel , i d=ÿ1, labe l="on / off " pos=(10 ,28 ) , size=wx . D efa   ,
                                     ul t Si ze )
        self . onOff . Bind (wx .EVT TOGGLEBUTTON, self .handleAudio )

        # Menu for choosing the effect
        fxs = [ 'Delay' Di sto 'Chorus' 'Reverb' ]
        self . popupText = wx. St a ti c T ext ( self . panel , i d=ÿ1,
                       label=" C h ose an effect          , pos = (10 ,60) , i    size=wx . D efa ul t Si ze )
        self . popup=wx. Choice (self.panel,          d=ÿ1, pos =(8 ,78) ,
                                size=wx . D efa ul t Si ze   ,   choices=fxs )
        self . popup . Bind (wx .EVT CHOICE, self .changeFx )
```

```
89              # Parameter 1
90              p 1l ab el= " delay , , depth orsize: 0. 5 0 0 self . p 1 text = wx . S
91              ta ti c T ext ( self . panel , i d=ÿ1, label=p 1l ab el pos = (140 ,10) , size =(250 ,20 ) )
92
93              self . p1 = wx. Slider ( self . panel , i d=ÿ1, v al ue =500, minValue=0,
94                                       maxValue=1000, pos =(140 ,32 ) , size =(250 ,20 ) )
95              self . p1. Bind (wx .EVT SLIDER, self .changeP1 )
96
97              # Parameter 2
98              p 2l ab el = "feed sl opefeedor damp: 0. 5 0 0 self . p 2 text =
99              wx . S ta ti c T ext ( self . panel , i d=ÿ1, label=p 2l ab el pos =(140 ,60) , size = (250 ,20 ) )
100
101             self . p2 = wx. Slider ( self . panel , i d=ÿ1, v al ue =500, minValue=0,
102                                      maxValue=1000, pos =(140 ,82 ) , size =(250 ,20 ) )
103             self . p2 . Bind (wx .EVT SLIDER, self .changeP2 )
104
105     def handleAudio(self Receive, event):
106             a ToggleEvent and start or stop the audio server. if evt. Get In t () == 1:
107
108                 s. start()
109             else :
110                 s. stop ( )
111
112     def changeFx(self , event):
113             Receives a ChoiceEvent and changes the effect heard. self . s wi
114             tcher . changeFx ( evt . Ge tS t ring ( ) )
115
116     def changeP1( self , event):
117             Receives an Sli from rE ventetmo di fi els parameter 1 of the heard effect.
118             x = event. Get Int ( ) ÿ 0. 0 0 1
119             self . p 1 text . S e tL ab el( "delay, drive , depth orsize: %.3 f " % x )
120             self.swi tcher.changeP1(x)
121
122     def changeP2(self , event):
123             Receives an Sli from rE ventetmo di fi els parameter 2 of the heard effect.
124             x = event. Get Int ( ) ÿ 0. 0 0 1
125             self . p 2 text . S e tL ab el( "feed , slip , feedor damp: %.3 f " % x )
126             self.swi tcher.changeP2(x)
127
128 # Edge object of the wx .App class (required)
129 app = wx . App ( )
130
131 # Sound source to process ( Change the sound source )
132 src= S fPl ayer ( "flute.aif" loop=True). mix(2). play ( )
133 # Effects manager
134 fxs witch = FxSwitch ( src )
135
136 # C rea ti on of the window . R ec oi tal ' argument 137 mainFrame ' swi cher ' an FxSwitch object
= MyFrame ( s wi tcher=fxs wi tch )
138 mainFrame . Show ( )
139 # Taking control of the execu ti on loop by wxPython
140 app. MainLoop()
```

scripts/chapter 13/10 fx switch.py

13.3 FM Synthesizer with Midi Control and GUI

To use this script, you must have a Midi keyboard connected to your computer!

```
#! /usr/bin/env python
# enc odin g: utf ÿ8
import wx
from pyo import ÿ

# Displays available Midi interfaces
pmlistdevices ( )
# Default Midi Interface
   idef = pm getdef ul tinput ()
prints            Default Midi interface: %d\n" % idef

# If the default Midi interface is not the correct one
# replace idef " with your interface number
# in the method " se tMi di I npu tD e vi ce s = S erver
( )
sec. se tMi di I npu tD e vice ( 9 9 )
sec. boot()
sec. amp=0.5

class FmSynth:
      set init _ _ ( self ):
            self . no te = no tein ( scale =1)
            self . amp = Port( self . note [ " self . velocity time ] , risetime=.001 , falltime =1)
            ratio = SigTo ( value =.25 init =.25) , =.05 ,
            self . inde x = SigTo ( value =5, time =.05 init =5) ,
            self . india xLi ne = self . amp ÿ self . india x
            self . fm1 = FM( carrier= self. no te [ "pi tch" ], inde x= ratio= self . ra ti o ,
                        self.indexLine , mul= self . ampÿ 0 . 1). mix()
            self . fm2 = FM( carrier= self. note [ "pitch" ]ÿ.9 9 7 ratio= self.ra ti o , inde x= self.indexLine ,
                        mul= self.ampÿ0.1). mix() ,
            self . fm3 = FM( carrier= self . note [ "pitch" ] ÿ ratio= self .ra ti o 9 9 5 inde x= self.indexLine mul= self.ampÿ0.1). mix() ,
                                                                        ,
            self . fm4 = FM( carrier= self. note [ "pitch" ] ÿ 1.0 0 2 inde x= , ratio= self . ra ti o ,
                        self.indexLine mul= self.ampÿ0.1). mix()
            self . mix = Mix ([self. fm1 self. filt , self . fm2 , self . fm3 self . fm4 ] , voices=2)
            = Biquad (self. mix , freq=2000, q=1, type=0) . out ( )

class MyFrame (wx.Frame):
      def _ _ ( selfinit , parent=None titl e=u "FM synth" pos = (100 ,100 ) , size = (550 ,
                  ,300) , synth=None ):
            wx . Frame . init ( self , parent , id=ÿ1, title=title , pos=pos , size=size )
            self . synth = synth

            self . p anel = wx . Panel ( self )
            self . panel. SetBackgroundColour( "#DAD3D0" )

            self . onO f fText = wx . S ta ti c T ext ( self . panel , i d=ÿ1, labe l="Audio" pos =(28 ,10) ) ,

            self . onOff = wx . ToggleButton ( self . panel , i d=ÿ1, labe l="on / off " pos =(10 ,28 ) ) ,

            self . onOff . Bind (wx .EVT TOGGLEBUTTON, self .handleAudio )

```

```
52          filtertypes = [ 'Lowpass 'Highpass ' 'Bandpass ' self . popupText = wx.  ,  'Bandstop' ]
53          St a ti c T ext ( self . panel , i d=ÿ1,
54                                                  labe l="Filter type                  " , pos =(10 ,60 ) )
55          self . popup=wx. Choice (self.panel,              i d=ÿ1, pos =(8 ,78) ,
56                                              size=wx . D efa ul t Si ze    ,  choices= filtertypes )
57          self . popup . S and String Selection ( "Lowpass" )
58          self . popup . Bind (wx .EVT CHOICE, self . chang eFil te rT ype )
59
60          attlabel = "Attack time: 0. 0 0 1 sec self . tex tA tt = wx .          "
61          S ta ti c T ext ( self . panel , i d=ÿ1, label=attlabel pos = (140 ,10) , size = (250 ,20 ) )      ,
62
63          self . slider Att = wx . Slider ( self . panel , i d=ÿ1, value =1, minValue=1,
64                                          maxValue=2000, pos =(140 ,32 ) , size =(400 ,20 ) )
65          self . slider A tt . Bind (wx .EVT SLIDER, self .changeAttack )
66
67          rellabel = "Release time: 1. 0 0 0 sec self . text R el = wx .          "
68          S ta ti c T ext ( self . panel , i d=ÿ1, label=rellabel pos = (140 ,60 ) , size = (250 ,20 ) )      ,
69
70          self . slider R el = wx . Slider ( self . panel , i d=ÿ1, v al ue =1000, minValue=1,
71                                          maxValue=10000, pos=(140,82), size=(400,20))
72          self . slider Real. Bind (wx .EVT SLIDER, self .ch an geRele a se )
73
74          ratlabel= "FM R a ti o: 0. 2 5 0 0 self . text          "
75          R a ti o = wx . S ta ti c T ext ( self . panel , i d=ÿ1, label=ratlabel pos = (140 ,110 ) , size =      ,
76                                          (250 ,20 ) )
77          self . slider Ratio = wx . Slider(self.panel, maxValue=2000,  i d=ÿ1, value =250, minValue=0,
78                                          pos=(140,132),                  size = (400 ,20) )
79          self . R atio slider. Bind (wx .EVT SLIDER, self .ch an geRa ti o )
80
81          indlabel= "FM Index: 5.0 0 0          "
82          self . text Index = wx . S ta ti c T ext ( self . panel , i d=ÿ1, label=in dl ab el pos = (140 ,160 ) ,  ,
83                                          size = (250 ,20 ) )
84          self . slider Index = wx . Slider(self.panel,              i d=ÿ1, value =5000, minValue=0,
85                                          maxValue=50000    , pos = (140 ,182 ) ,      size = (400 ,20 ) )
86          self . I ndex slider. Bind (wx .EVT SLIDER, self .change Index )
87
88          filtlabel= "Filter frequency: 2000 Hz"
89          self . tex tF req = wx . S ta ti c T ext ( self . panel , i d=ÿ1, label= filtlabel pos = (140 ,210 ) ,     ,
90                                          size = (250 ,20 ) )
91          self . slider F req = wx . Slider ( self . panel ,              i d=ÿ1, value =2000, minValue =100,
92                                          maxValue=10000    , pos = (140 ,232 ) ,      size = (400 ,20 ) )
93          self . slider Freq. Bind (wx .EVT SLIDER, self .changeFreq )
94
95      def handleAudio(selfevt):      ,
96          if event . Get In t ( ) == 1:
97              s. start()
98          else :
99              s. stop ( )
100
101     def chang eFil te rT ype ( self typ =   ,  event):
102         evt . Ge t S t ring ( )          ,
103         if type ==          Low pass    :
104             self. synth. filt . type = 0
105         elif typ== "Highpass"              :
```

```
106                     self . synth. filt . type = 1
107                 elif typ == "Bandpass" :
108                     self . synth. filt . type = 2
109                 elif typ== "Bandstop" :
110                     self . synth. filt . type = 3
111
112         def changeAttack(selfevt):      ,
113             x = event. Get Int ( ) ÿ 0. 0 0 1
114             self . tex tA tt . S e tL ab el ( "Attack time: %.3 fsec " % x )
115             self . synth. amp. rise time = x
116
117         def ch an geRele a se ( selfevt );
118             x = event. Get Int ( ) ÿ 0. 0 0 1
119             self . text R el . S e tL ab el ( "Release time: %.3 fsec " % x )
120             self . synth. amp. falltime = x
121
122         def changeR a tio ( selfevt ): ,
123             x = event. Get Int ( ) ÿ 0. 0 0 1
124             self . text R a ti o . S e tL ab el ( "FM R a ti o : %.3 f " % x )
125             self . synth. ratio . value = x
126
127         def change Index ( selfevt ):  ,
128             x = event. Get Int ( ) ÿ 0. 0 0 1
129             self . text I ndex . S e tL ab el ( "FM Index: %.3 f " % x )
130             self . synth. india x . value = x
131
132         def changeFreq(selfx =      ,   event):
133             evt. Get Int()
134             self . tex tF req . S e tL ab el ( "Filter frequency: %d Hz" % x )
135             self . synth. filt . freq = x
136
137 app=wx. App ( )
138
139 synth = FmSynth()
140 mainFrame = MyFrame ( synth=synth )
141 mainFrame . Show ( )
142
143 app. MainLoop()
```

scripts/chapter 13/11 fm_synth.py

Chapter 14

Creating GUIs 2

14.1 2-dimensional control interface

14.1.1 Preamble

The project of the day consists in modifying the script which makes it possible to alternate the effect applied on a sound source (scripts/chapter 13/10 fx switch.py) to give it a bit more flexibility.
We will first see how to display a standard dialog to select the sound at source of the effect then, in a second step, we will design a small 2-dimensional controller which will make manipulating the parameters more interesting.

The sound class

Let's first establish our audio class based on the class FxSwitch, slightly modified. The object created from the FxSwitch class can be manipulated via the interpreter extension in the server window.

```
1 #! /usr/bin/env python
2 # enc odin g: utf ÿ8
3
4 from pyo import ÿ
5
6s = Server ( ) . boot()
7 sec. amp=0.5
8
9 class FxSwitch:
10          def  _ _ initiate _ _ (selffx=', Delay' ):
11                  # keep a referenceal ' effect
12                   self . fx = fx
13                  # listec on ten an t twice the same mono sound to create a tablestereo
14                   self . table = SndTable ([SNDS PATH+"/ transparent.aif" ] ÿ 2)
15                  # reading the table in a loop
16                   self . input = Osc ( self . ta bl e # input    ,    freq= self . your blue. get ( ) )
17                  of 2 parameters with a portamento of 50 ms .
18                   self . p1 = SigTo ( value =0.5 init =0.5,  time =0.05     ,
19                   self . p2 = SigTo ( value =0.5 init =0.5,  time=0.05      ,
20
```

```
21              # the 5 effects available .
22              # Note the call. stop() to disable sal' init processes
23              self . d el ay = Delay (self. input , feedback= self. p2 , mul=.5) . stop ( ) ,
24              self . distortion = Di sto (self. input , sl ope= self. p2 , mul=.5) . stop ( ) ,
25              self . de grade = Degrade ( self.input , srscal bi tdept h=Si g ( self . p1 , mul=20, add=2) ,
26                                          e=Si g (self.p2, mul=.99 mul=1.5).stop() , add=.01) ,
27
28              self . reverb = WGVerb(self.input , feedback= self.p1 ,
29                                     cutof f=Si g (self. p2, mul=10000 self. , add=250) ) . stop ( )
30              harmo = Harmonizer (self. input ,
31                                  transp o=Si g ( self . p1 , mul=24, add=ÿ12) ,
32                                  feedback= self . p2). stop ( )
33              # dictionary to point to the object of the effectc or r an t ,
34              self . fxdict = { 'Delay' : self. delay, Di sto : self. di sto : self . degrade, : self. reve ,
35                                rb 'Degrade' 'Reverb' : self. harmony} ,
36                                harmonizer
37              # call the out() method on the given effect argument
38              self . fxdict [fx]. out ( )
39
40      def changeFx ( selffx ):  ,
41              # A rretel 'effectcor r an t
42              self . fxdict [self. fx]. stop ( )
43              # Start the new effect
44              self . fxdict [fx]. out ( )
45              # Replace the active referenceal ' effect
46              self . fx = fx
47
48      def changeTable(self , snd ):
49              # place a new sound in the table
50              self . table. sound=snd
51              # adjust the playback speed according to the length of the new sound
52              self . input . freq = self. your blue. get tRa te ( )
53
54      def changeP1 ( selfx ):  ,
55              self . p1. value = x
56
57      def changeP2 ( selfx ):  ,
58              self . p2 . value = x
59
60 fxs witch = FxSwitch()
61
62 sec. gui( locals ())
63
```

scripts/chapter 01/14 class fxswitch.py

The basic GUI

The next step is to add a basic GUI, i.e. an object application (wx.App), a main window (wx.Frame), a panel (wx.Panel), a button to control server activation (wx.ToggleButton) and a menu to change the effect heard (wx.Choice).

```
#! /usr/bin/env python
# enc odin g: utf ÿ8
import wx
from pyo import ÿ

s = Server ( ) . boot()
sec. amp=0.5

class FxSwitch:
        def _ _ initiate _ _ , fx=' Delay ' ):
                (selfself.fx = fx
                self . table = SndTable ([SNDS PATH+"/ transparent.aif" ] ÿ 2)
                self . input ut = Osc (self .ta bl efreq= self .ta bl e .ge tR a te ( ) )
                self . p1 = SigTo ( value =0.5 init =0.5) , time =0.05 ,
                self . p2 = SigTo ( value =0.5 init =0.5) , time=0.05 ,
                self . d el ay = Delay ( self.input , d el ay= self.p1, feedback= self.p2). stop ( )
                self . distortion = Di sto ( self . input , drift= self . p1 , sl ope= self . p2 ,
                                        mul=.5) . stop ( )
                self . de grade = Degrade ( self . input , bi tdept h=Si g ( self . p1 , mul=20, add=2) ,
                                        srscal e=Si g ( self . p2 , mul=.99 mul=1.5) . , add=.01) ,
                                        stop ( )
                self . reverb = WGVerb(self.input , feedback= self.p1 ,
                                        cutof f=If g (self.p2, mul=10000, add=250) ) . stop ( )
                self . harmo = Harmonizer ( self.input ,
                                        transp o=Si g ( self . p1 , mul=24, add=ÿ12) ,
                                        feedback= self . p2). stop ( ) ,
                self . fxdict = { 'Delay' : self. delay, Di sto : self. di sto 'Degrade' : self. degraded , ,
                                        'Reverb' : self. dream rb ,
                                        ' harmonizer ' :self. harmony}
                self . fxdict [fx]. out ( )

        def changeFx ( selffx ): ,
                self . fxdict [self. fx]. stop ( )
                self . fxdict [fx]. out ( )
                self . fx = fx

        def changeTable(self snd): ,
                self . table. sound=snd
                self . input . freq = self. your blue. get tRa te ( )

        def changeP1 ( selfx ): ,
                self . p1. value = x

        def changeP2 ( selfx ): ,
                self . p2 . value = x

# creation of the main container in front of the wx class. frame
class MyFrame (wx.Frame):
        def init (self , parent=None titl e="Fx Swi tcher size = (600 ,500) , s wi " , pos = (100 ,100 ) ,
                        tcher=None ):
                wx. Frame . init ( self i d=ÿ1, title=title , parent , , pos=pos , size=size )

                # referenceal ' FxSwitch object that one desires to control
                self . s wi tcher = s wi tcher
```

```
55
56                  # panel where to place the controls
57                  self . p anel = wx . Panel ( self )
58                  self . panel. SetBackgroundColour( "#DDDDDD" )
59
60                  # Toggle to control server activation
61                  self . onO f fText = wx . S ta ti c T ext ( self . panel , i d=ÿ1, labe l="Audio" pos = (28 ,10) )
62
63                  self . onOff = wx . ToggleButton ( self . panel , i d=ÿ1, labe l="on / off " pos =(10 ,28) )
64
65                  # an action on the toggle calls the self method. handleAudio
66                  self . onOff . Bind (wx .EVT TOGGLEBUTTON, self .handleAudio )
67
68                  # Menu for choosing the effect
69                  fxs = [ 'Delay' Di sto 'Degrade' self . popupText = 'Reverb' 'harmonizer' ]
70                  wx. St a ti c T ext ( self . panel , i d=ÿ1,
71                                                  labe l="Choose FX" , pos = (10 ,100) )
72                  self . popup=wx. Choice (self.panel, i d=ÿ1, pos=(8,118), choices=fxs)
73                  # an action on the menu calls the self method. changeFx
74                  self . popup . Bind (wx .EVT CHOICE, self .changeFx )
75
76          def handleAudio(selfevt):
77                  if event . Get In t ( ) == 1:
78                          s. start()
79                  else :
80                          s. stop ( )
81
82          def changeFx ( self # , event):
83                  evt . Ge tS t ring ( ) returns the text of the chosen element
84                  self . s wi tcher . changeFx ( evt . Ge tS t ring ( ) )
85
86 app = wx . App ( )
87
88 fxs witch = FxSwitch()
89
90 mainFrame = MyFrame ( s wi tcher=fxs wi tch )
91 mainFrame . Show ( )
92
93 app. MainLoop()
```

scripts/chapter 02/14 base frame.py

14.1.2 Standard Button and Dialog

We will now include 2 new features that will allow the user to navigate through its own file hierarchy to choose the sound that will be used as the source in the effects.

The wx.Button object allows to add a button in a panel with possibility to specify the text written on the button itself. The operation of the button is very simple, at each when clicked, it sends a wx.EVT BUTTON event. The method related to this event will cause a standard file selection dialog to appear.

We create a dialog with the wx.FileDialog object whose ShowModal() method will cause the screen display. The behavior of this object is particular since the display of the dialog blocks the execution of the code until the ShowModal() method returns the result operations (Open, Save, Cancel, etc.). The code following the line where the dialog is displayed will therefore be executed only after having clicked on one of the buttons offered in the window, which allows time for the user to choose his file before the method processes the information.
When initializing the dialog, it is possible to specify the message for the user in the message argument. The arguments defaultDir and defaultFile allow to respectively specify the folder where the dialog points at the opening as well as a default file name (for save dialog).

Some WxPython constants concerning dialogs

- wx.FD OPEN given to the style argument of the wx.FileDialog object configures the latter for file selection.
- wx.FD SAVE given to the style argument of the wx.FileDialog object configures the latter to file backup.
- wx.ID OK is a numeric constant, returned by the dialogs, corresponding to the "Open" and "Save" buttons.
- wx.ID CANCEL is a numeric constant, returned by dialogs, corresponding to the "Cancel" button.

When the user has made his choice, the "path" of the chosen file can be retrieved, in format string, using the dialog's GetPath() method. After having collected the useful information, do not forget to call the Destroy() method on the dialog in order to destroy it as well as its resources.

The wildcard argument of the wx.FileDialog object is very useful for filtering out the files that the user can select. A string specifying the type of file as well as its extension must be provided. Here is an example to allow only AIFF files:

```
wil dcard= "AIFF(ÿ.aif) | ÿ . aif"
```

We separate, with a vertical bar, the message to be displayed in the menu (AIFF (*.aif)) from the list of filtered files (*.aif), the star signifying "all" files ending in .aif".

You can give more than one choice of file type, by adding "type|extensions" pairs, as well as several possible extensions for a given type, separated by a semicolon.

```
wildcard=  " All files | ÿ  · ÿ | AIFF | ÿ .aif; ÿ .aiff | WAV | ÿ . if; ÿ . aiff; ÿ .  . wav; ÿ
```

The \ symbol continues a string on the next line. Here is a more version elegant!

```
wildcard=       " All files | ÿ "AIFF · ÿ | " \
                file | ÿ "Wave file   . if; ÿ . aiff; ÿ . aifc; ÿ . AIF; ÿ . AIFF; ÿ . Aif; ÿ . Aiff | " \
                | ÿ . wav; ÿ                    . wave; *.WAV; ÿ.WAVE; ÿ.Wav; ÿ .Wave"
```

Here is our modified program:

```
#! /usr/bin/env python
# enc odin g: utf ÿ8
import wx
from pyo import ÿ

s = Server ( ) . boot()
sec. amp=0.5

class FxSwitch:
        def    ¯ ¯ initiate ¯ ¯          ,   fx=' Delay ' ):
               (selfself.fx = fx
               self . table = SndTable ([SNDS PATH+"/ transparent.aif" ] ÿ 2)
               self . input ut = Osc (self . ta bl efreq= self . ta bl e . ge tRa te ( ) )
               self . p1 = SigTo ( value =0.5 init =0.5,  time =0.05      ,
               self . p2 = SigTo ( value =0.5 init =0.5,  time=0.05       ,
               self . d el ay = Delay (self. input , feedback= self.p2 ) . stop (),
               self. distortion = Di sto ( self . input , drift= self.p1 , sl ope= self .
                                          mul=.5) . stop ( )
               self . de grade = Degrade ( self.input , srscal e=Si    bi tdept h=Si g ( self . p1 , mul=20, add=2) ,
                                          g (self.p2, mul=.99 mul=1.5).stop()        , add=.01) ,

               self . reverb = WGVerb(self.input , feedback= self.p1 ,
                                          cutof f=Si g (self. p2, mul=10000 self.       , add=250) ) . stop ( )
               harmo = Harmonizer (self. input ,
                                          transp o=Si g ( self . p1 , mul=24, add=ÿ12) ,
                                          feedback= self . p2), stop ( ) ,
               self . fxdict = { 'Delay' : self. delay, Di sto : self. di sto 'Degrade' : self. degrade,        ,
                                 'Reverb' : self. dream rb: self. harmony}                                    ,
                                  Harmonizer
               self. fxdict [fx]. out ( )

        def                        ,   fx):
            changeFx(selfself.fxdict[self.fx].stop()
               self . fxdict [fx]. out ( )
               self . fx = fx

        def changeTable(self snd):     ,
               self . table. sound=snd
               self . input . freq = self. your blue. get tRa te ( )

        def changeP1 ( selfx ):    ,
               self . p1. value = x

        def changeP2 ( selfx ):    ,
               self . p2 . value = x

class MyFrame (wx.Frame):
        def ( selftitle="Tu Swi tcher init , parent=None, size = (600 ,500 ) , s wi       , pos = (100 ,100 ) ,
```

```
50              wx. Frame . __ initiate __ (self, parent,  i d=ÿ1, title=title  , pos=pos ,  size=size )
51
52              self . s wi tcher = s wi tcher
53              self . p anel = wx . Panel ( self )
54              self . panel. SetBackgroundColour( "#DDDDDD" )
55
56              self . onO f fText = wx . S ta ti c T ext ( self . panel , i d=ÿ1, labe l="Audio" pos =(28 ,10) )  ,
57
58              self . onOff = wx . ToggleButton ( self . panel , i d=ÿ1, labe l="on / off " pos =(10 ,28 ) )  ,
59
60              self . onOff . Bind (wx .EVT TOGGLEBUTTON, self .handleAudio )
61
62          # Button to open a dialog and select a new sound  "
63              self . choose Bu t ton = wx . Button ( self . panel , i d=ÿ1, labe l="Load snd . . . pos =(10 ,65) )  ,
64
65          # an action on the button calls the self method. loadSnd
66              self . chooseBu t your . Bind (wx .EVT BUTTON, self .loadSnd )
67
68              fxs = [ 'Delay' 'Degrade' 'Harmonizer' self . popupText = wx. St a t i c T ext ( self . panel , i  ' ]
69              d=ÿ1,
70                                                  labe l="Choose FX" , pos = (10 ,100) )
71              self . popup=wx. Choice (self.panel, i d=ÿ1, pos=(8,118), choices=fxs)
72              self . popup . Bind (wx .EVT CHOICE, self .changeFx )
73
74      def handleAudio(selfevt):  ,
75              if event . Get Int ( ) == 1:
76                  s. start()
77              else :
78                  s. stop ( )
79
80      def loadSnd(selfevt):  ,
81          # filter available files
82              wil dcard= "AIFF  " All files | ÿ ÿ | " \
83                         file | ÿ . Aiff | " \  . if; ÿ . aiff; ÿ . aifc; ÿ . AIF; ÿ . AIFF; ÿ . Aif; ÿ "Wave file | ÿ .
84                         wave; *.WAV; ÿ.WAVE; ÿ.Wav; ÿ .Wave"
85          # creation of a di al o guest and a rd for the selection of a file  "
86              dl g = wx . F il e D ialog ( self , message="Choose a new soundfile . . . style=wx .FD OPEN)  ,
87                                  wil dcard=wildc a rd  ,  -
88          # the ShowModal( ) method displays the current window in the foreground and
89          # blocks execution until the if dl g . ShowModal() == wx. OK  user presses "OK" or "Cancel "
90              ID:  -
91                  # get the path of the selected file
92                  path = dlg. GetPath()
93                  if path != "" # call  :
94                      method changeTable with new sound as argument
95                      self . s wi tcher . changeTable ( path )
96          # when everything is finished we destroy the di al ogue
97              dl g. Troy ( )
98
99      def changeFx(selfevt):  ,
100             self . s wi tcher . changeFx ( evt . Ge tS t ring ( ) )
101
102 app = wx . App ( )
103 fxs wi tch = FxSwitch()
```

```
mainFrame = MyFrame ( s wi tcher=fxs wi tch )
mainFrame . Show ( )
app. MainLoop()
```

scripts/chapter 14/03 button dialogue.py

14.1.3 2-dimensional control surface

Here we are at the stage of implementing a control surface allowing us to modify the 2 parameters of the effects of a single movement. Using a panel and a drawing surface, we will use the mouse position on the abscissa to control the first effect and the position in ordinate to control the second.

Let's first start by creating a new class (Surface) derived from the wx.Panel class. This step is necessary since you need a container independent of the main container in order to specify the dimensions of the drawing area.

```
class Surface (wx.Panel):
    def init(selfsize):_ , parent , pos , wx .
        Panel . init(self, parent, pos=pos, size=size )
```

Several events will be considered in this class. First, an essential event for any program that draws shapes on the screen, wx.EVT PAINT. This event occurs each time the panel is asked to refresh its visual content, by calling its self.Refresh() method. Thus, each time one wishes to draw on the screen, one must cause a wx.EVT PAINT event.

Secondly, we want to be informed about the different movements of the mouse on the panel. The wx.EVT LEFT DOWN events, when the left button of the mouse is pressed, wx.EVT LEFT UP, when the left mouse button is released and wx.EVT MOTION, when the mouse moves will each be assigned a method to inside which we will organize our drawing.

```
class Surface (wx.Panel):
    def init(selfsize):_ , parent , pos , wx .
        Panel . init(self, parent, pos=pos, size=size )

        # If the mouse is self .  is not in f on cee  ,  position is None ( easy to filter )
         pos = None
        # P o si ti on of the circle representing the current position
         self . circle Pos = ( 2 0 0  ,2 0 0 )

        # wx .EVT PAINT is sent by the self method. Refresh ( )
         self . Bind(wx.EVT PAINT, self.OnPaint)
        # left mouse button pressed
         self . Bind (wx .EVT LEFT DOWN, self .OnMouseDown )
        # left mouse button released
         self . Bind (wx .EVT LEFT UP, self .OnMouseUp )
        # move the mouse over the surface
         self . Bind (wx .EVT MOTION, self .OnMotion )
```

Let us observe in detail the behavior of our different methods.

OnMouseDown

First, we want to capture the mouse, that is, when we press the mouse button left while being above the panel, we prohibit the interaction with the other elements graphics to avoid conflicts. Then, we keep the position of the pointer in memory in order to to know where the mouse is at the drawing stage. Finally, we ask to refresh the screen.

```
def OnMouseDown(selfevt):  ,
      # CaptureMouse submits the mouse ' object while
      # ReleaseMouse() is not called
       self . CaptureMouse ( )
      # event. G e tP o si ti on ( ) returns the position of the pointer in pixel (x , y )
       self . pos = evt. G e tPo si ti on ( )
      #self. Refresh ( ) sends a wx .EVT PAINT event to refresh ' self . Refresh ( )                    screen
```

OnMouseUp

This method has a role to play only if the mouse has been pressed above the panel, that is, self.CaptureMouse() has been called. If this is indeed the case, we release the mouse, we reset the position and refresh the screen.

```
def OnMouseUp(selfevt):  ,
      #self. HasCapture() returns True if if self.              the mouse is oblivious to the object
       HasCapture():
            # releasethemouse
             self . ReleaseMouse()
            # resetposition
             self . pos = None
            # refreshes self.        screen
             Refresh ( )
```

OnMotion

In this method, we keep track of the movements of the pointer, always if the panel has well taken control of the mouse, and we refresh the screen systematically. Be careful that the pointer does not send values outside the limits of the panel!

```
def OnMotion(selfevt):    ,
      "OnMotion is called each time the mouse is placed on the panel"
       if self . HasCapture():
            # panel size, w, h = self. Ge    you like (X, Y)
            tSize ( )
            # current position of the sou ri s self . pos =      ,   you like (X, Y)
             evt. G e tPo si ti on ( )
```

```
        # limit the position value between 0 and the size in X or in Y
        if self . pos [ 0 ] < 0:
            self . pos[ 0 ] = 0
        elif self . pos[ 0 ] > w:
            self . pos[0] = w
        if self . pos [ 1 ] < 0:
            self . pos[ 1 ] = 0
        elif self . pos[ 1 ] > h:
            self . pos[ 1 ] = h
        # refreshes self.          screen
        Refresh ( )
```

OnPaint

It is in this method, called each time the screen is refreshed, that we draw shapes on the panel. To better illustrate the position of the pointer, we will draw a large cross, one line the full height and one the full width, the crossing of which is will do at the exact location where the pointer is. For fantasy, we will add a red circle around the center point.

First rule

We always draw on an object derived from the class wx.DC, which, in the case of a drawing caused by a wx.EVT PAINT, must absolutely be a wx.PaintDC or wx.AutoBufferedPaintDC object. . The latter is preferable since it allows a smoother screen refresh under Windows and linux. However, it requires calling the SetBackgroundStyle method on our panel with wx.BG STYLE CUSTOM as argument.

```
class Surface (wx.Panel):
    def init . parent, pos=wx.DefaultPosition, size ,
        wx. Panel . init, parent, pos=pos, self. Se tB ack gr oundS
        tyle (wx .BG STYLE CUSTOM)
```

Second rule

A wx.PaintDC object must always be recreated each time the method is called, so we do not keep a class reference for this object (the variable does not start with self).

Two essential drawing tools are the brush (wx.Brush) and the pencil (wx.Pen). The first allows to specify the color used to fill the shapes while the second determines the color and thickness of the contour lines.

Arguments when initializing the wx.Brush object

- color: Color of the brush (specify in hexadecimal or using a wx.Color object).
- style: Style of the brush (solid by default). See the manual for the different styles possible.

Arguments when initializing the wx.Pen object

- color: Color of the pencil (specify in hexadecimal or using a wx.Color object).
- width: Thickness of the line in pixels.
- style: Pencil style (solid by default). The list of possible styles is accessible through the wxwidgets documentation.

In order, the following method draws a rectangle to the dimensions of the panel in order to control the background color and a grid every 50 pixels. Then, if the position is not None, we draw two lines that intersect at the position of the pointer then, at the very end, we draws a circle centered on the last known position and displays the coordinates of the pointer. Note the color change of the brush before drawing the circle!

```
def OnPaint(selfevt):
    w, h = self. Ge tSize ( )
    # we draw on a wx object. PaintDC
    dc=wx. AutoBufferedPaintDC ( self )
    # specify the color of the current brush ( for the interior of sform s )
    dc. SetBrush(wx.Brush( "#444444" ))
    # draw a rectangle with the dimensions of the panel
    dc. DrawRectangle(00,w,h),
    # specify the color of the pencil for the framing
    dc. SetPen (wx . Pen ( "#666666" , #  1 ) )
    we draw a grid every 50 pixels . . .
    for i inr an ge ( 0 4 0 0 dc . , 5 0 ):
        DrawLine ( 0 dc . , I , w, i )
        DrawLine ( i # specify 0 i, , h)
    the pen color for the cross
    dc. SetPen(wx.Pen( "#AAAAAA" , 1))
    # ifself . pos is not None we draw a cross at the current position
    if self . pos != None:
        # adjust the position of the circle
        self . circle Pos = self . pos
        # draw a line horizontalalah at the teu r Y
        dc. DrawLine(0 choke.pos[1], w, choke.pos[1])
        # draw a vertical line a wide X
        dc. DrawLine(self.pos[0] self.pos[0],h)0 ,
        # convert if we have positions X and Y between 0 and 1
        x = self. pos[0]/ float (w)
        y=1. #  ÿ self . pos[1]/ float (h)
        displays the normal position of the pointer
        dc. DrawText( "%.3f%.3f" %(x, y), 1 0 # callback call , 1 0 )
        to route values to audio process
        self . callback(x,y)
    # new brush eco ul for the circle
    dc. SetBrush(wx.Brush( "#AA0000" ))
    # draw a circle centered on the current position
    dc. DrawCircle(self.circle Pos[0] self.circle Pos[1] , 5 )
```

You will probably have noticed the conversion of position values between 0 and 1 and then the call a method called self.callback. This method, which does not yet exist, will be given to

the initialization of the Surface object by our main window. It is by this method that we We'll route the values to the object that generates the audio process. The init method of our class therefore varies slightly:

```
class Surface (wx.Panel):
    def (selfcallback):
        wx. Panel . init, parent, pos=pos, self. Se tB ack gr oundS
        tyle (wx .BG STYLE CUSTOM)
        self . callback = callback
        self . pos = None
        . . .
        . . .
```

We now need to add a method to our MyFrame class. This method must receive 2 arguments, one for the normalized position in X and one for the normalized position in Y. Its only purpose is to transfer the values to our audio object:

```
def changeParams(self, y):
    # function called by self . s wi tcher .
    changeP1(x)
    self . s wi tcher . changeP2(y)
```

All that remains is to create a Surface object in our main window. At the very end of the init method of the MyFrame class, we add the following line:

```
self . area = Area(self.panel, pos=(150,28), size=(400,400),
                   callback= self . changeParams )
```

* Note the passing as argument of the reference (and not the call because there is no parentheses) to the self.changeParams method. It is the self.surface object that will take care of the calls.

Here is our program with the addition of the control surface:

```
1 #! /usr/bin/env python
2 # enc odin g: utf ÿ8
3 import wx
4 from pyo import ÿ
5
6s = Server ( ) . boot()
7 sec. amp=0.5
8
9 class FxSwitch:
10      def
11          (selfself.fx = fx
12          self . table = SndTable ([SNDS PATH+"/transparent.aif" ] ÿ 2)
13          self . input ut = Osc (self . ta bl efreq= self . ta bl e . ge tRa te ( ) )
14          self . p1 = SigTo ( value =0.5 init =0.5)
15          self . p2 = SigTo ( value =0.5 init =0.5)
16          self . d el ay = Delay ( self.input , d el ay= self.p1, feedback= self.p2). stop ( )
```

```
        self . distortion = Di sto ( self.input , mul=.5) .  d rive = self . p1 ,        slope= self . p2 ,
                                    stop ( )
        self . de grade = Degrade ( self . input , bi tdept h=Si g ( self . p1 , mul=20, add=2) ,
                                        srscal e=Si g ( self . p2 , mul=.99          , add=.01) ,
                                        mul=1.5) . stop ( )
        self . reverb = WGVerb(self.input , feedback= self.p1 ,
                                    cutof f=Si g (self. p2, mul=10000 self.       , add=250) ) . stop ( )
        harmo = Harmonizer (self. input ,
                                        transp o=Si g ( self . p1 , mul=24, add=ÿ12) ,
                                        feedback= self . p2), stop ( ) ,
        self . fxdict = { 'Delay' : self. delay, Di sto : self. di sto 'Degrade' : self. degrade, :        ,
                                  self. harmony}  ,                'Reverb' : self. dream rb           ,
                                   harmonizer
        self . fxdict [fx]. out ( )

    def changeFx ( selffx ):    ,
        self . fxdict [self. fx]. stop ( )
        self . fxdict [fx]. out ( )
        self . fx = fx

    def changeTable(self snd):     ,
        self . table. sound=snd
        self . input . freq = self. your blue. get tRa te ( )

    def changeP1 ( selfx ):     ,
        self . p1. value = x

    def changeP2 ( selfx ):     ,
        self . p2 . value = x

# Spar am e ter control surface
# The class is a side of wx . Panel since # you have to draw on any         ' he
surface!
class Surface (wx . Panel ):
    def _ _ init, parent(selfpos, callback)size=size)          size   ,
        wx. Panel . init, parent, pos=pos, self. Se tB ack gr oundS
        tyle (wx .BG STYLE CUSTOM)          _          _

        # referenceal ' argument# when we   " callback"        which is the function of app el er
        place it on the surface
        self . callback = callback
        # If the mouse is self .     is not in f on cee         ,  position is None ( easy to filter )
        pos = None
        # P o si ti on of the circle representing the current position
        self . circle Pos = ( 2 0 0          , 2 0 0 )

        ### assignment of methods to certain events ###
        # wx .EVT PAINT is sent by the self method. Refresh ( )
        self . Bind(wx.EVT PAINT, self.OnPaint)
        # left mouse button pressed
        self . Bind (wx .EVT LEFT DOWN, self .OnMouseDown )
        # left mouse button relsche
        self . Bind (wx .EVT LEFT UP, self .OnMouseUp )
        # move the mouse over the surface
```

```
71              self . Bind (wx .EVT MOTION, self .OnMotion )
72
73      def OnMouseDown(selfevt):  ,
74          # CaptureMouse submits the mouse ' object while
75          #ReleaseMouse() n self.      is not called el ee
76           CaptureMouse ( )
77          # event. G e tP o si ti on ( ) returns the position of the pointer in pixel (x , y )
78           self . pos = evt. G e tPo si ti on ( )
79          #self. Refresh ( ) sends a wx .EVT PAINT event to refresh ' self . Refresh ( )                screen
80
81
82      def OnMouseUp(selfevt):  ,
83          #self. HasCapture() returns True if if self.                the mouse is oblivious to the object
84           HasCapture():
85                  # releasethemouse
86                   self . ReleaseMouse()
87                  # resetposition
88                   self . pos = None
89                  # refreshes self.        screen
90                   Refresh ( )
91
92      def OnMotion(selfevt):    ,
93          "OnMotion is called each time the mouse is placed on the panel"
94           if self . HasCapture():
95                  # panel size, w, h = self. Ge     you like (X, Y)
96                  tSize ( )
97                  # current position of the sou ri s self . pos =      ,   you like (X, Y)
98                   evt. G e tPo si ti on ( )
99                  # limit the position value between 0 and the size in X or in Y
100                  if self . pos [ 0 ] < 0:
101                         self . pos[ 0 ] = 0
102                  elif self . pos[ 0 ] > w:
103                         self . pos[0] = w
104                  if self . pos [ 1 ] < 0:
105                         self . pos[ 1 ] = 0
106                  elif self . pos[ 1 ] > h:
107                         self . pos[ 1 ] = h
108                 # refreshes self.        screen
109                  Refresh ( )
110
111     def OnPaint(selfevt):    ,
112         w, h = self. Ge tSize ( )
113         # we draw on a wx object. PaintDC
114          dc=wx. AutoBufferedPaintDC ( self )
115         # specify the color of the current brush ( inside the shape s )
116          dc. SetBrush(wx.Brush( "#444444" ))
117         # draw a rectangle with the dimensions of the panel
118          dc. DrawRectangle(0,w,h) ,  0
119         # specify the color of the pencil for the framing
120          dc. SetPen (wx . Pen ( "#666666" , #      1 ) )
121         we draw a grid every 50 pixels . . .
122          for i inrange ( 0 4 0 0 dc, .       ,  5 0 ):
123                 DrawLine ( 0 i dc .  ,    , w, i )
124                 DrawLine ( i          ,  0 h )  ,
```

```
125             # specify the color of the pencil for the cross
126             dc. SetPen(wx.Pen( "#AAAAAA" , 1 ) )
127             #siself.pos n if self.pos != None: one we draw a cross at the current position
128
129                 # adjust the position of the circle
130                 self . circle Pos = self . pos
131                 # draw a line horizontalalah at the teu r Y
132                 dc. DrawLine(0 choke.pos[1], w, choke.pos[1])
133                 # draw a vertical line a wide X
134                 dc. DrawLine(self.pos[0], self.pos[0], 0),
135                 # convert if we have positions X and Y between 0 and 1
136                 x = self. pos[0]/ float (w)
137                 y=1. #      ÿ self . pos[1]/ float (h)
138                 displays the normal position of the pointer
139                 dc. DrawText( "%.3f%.3f",%(x,y), 10 , 10 )
140                 # call the callback to route the values to the audi o process
141                 self . callback(x,y)
142             # new brush eco ul for the circle
143             dc. SetBrush(wx.Brush( "#AA0000" ))
144             # draw a circle centered on the current position
145             dc. DrawCircle(self.circle Pos[0] self.circle Pos[1] , 5 )
146
147 class MyFrame (wx.Frame):
148         def __init(self, parent=None titl e="Fx Swi tcher size = (600,500) , s wi ", pos = (100 ,100 ) ,
149                    tcher=None ):
150             wx. Frame . init_( self i d=ÿ1, title=titler, parent , self . s wi tcher = s wi , pos=pos , size=size )
151
152             self . p anel = wx . Panel ( self )
153             self . panel. SetBackgroundColour( "#DDDDDD" )
154
155             self . onO f fText = wx . S ta ti c T ext ( self . panel , i d=ÿ1, labe l="Audio" pos =(28 ,10) ) ,
156
157             self . onOff = wx . ToggleButton ( self . panel , i d=ÿ1, labe l="on / off " pos =(10 ,28 ) ) ,
158
159             self . onOff . Bind (wx .EVT TOGGLEBUTTON, self .handleAudio )
160
161             self . choose Bu t ton = wx . Button ( self . panel , i d=ÿ1, labe l="Load snd . . . pos =(10 ,65) ) ",
162
163             self . chooseBu t your . Bind (wx .EVT BUTTON, self .loadSnd )
164
165             fxs = [ 'Delay' 'Degrade' self . popupText = wx. St , 'Reverb' , ' harmonizer ' ]
166             a ti c T ext ( self . panel , i d=ÿ1,
167                                                 labe l="Choose FX" , pos = (10 ,100) )
168             self . popup=wx. Choice (self.panel, self.popup.S i d=ÿ1, pos =(8 ,118 ) , choices=fxs )
169             and Selection(0)
170             self . popup . Bind (wx .EVT CHOICE, self .changeFx )
171
172             # create a Surface object for the control of the parameters
173             self . area = Area(self.panel, pos=(150,28), size=(400,400),
174                                callback= self . changeParams )
175
176         def handleAudio(selfevt):      ,
177             if event . Get Int ( ) == 1:
178                 s. start()
```

```
179                 else :
180                     s. stop ( )
181
182         def loadSnd(selfevt):    ,
183             wildcard=          ” All files | ÿ          · ÿ | ” \
184                     ”AIFF file | ÿ . Aiff | ÿ . aiff; ÿ . aifc; ÿ . AIF; ÿ . AIFF; ÿ . Aif; ÿ ”Wave file
185                     | ÿ . wave; *.WAV; ÿ .WAV; ÿ .WAVE; ÿ.Wav; ÿ .Wave”                                    ”
186             dl g = wx . F il e D ialog ( self        ÿ , message=”Choose a new soundfile . . .                  ,
187                                         wil dcard=wildc a rd        ,   style=wx.FD OPEN)
188             if dl g . ShowModal() == wx. OK ID:     -
189                 path = dlg. GetPath()
190                 if path != ”” self .     :
191                     s wi tcher . changeTable ( path )
192             dl g. Troy ( )
193
194         def changeFx(selfevt):    ,
195             self . s wi tcher . changeFx ( evt . Ge tS t ring ( ) )
196
197         def changeParams(self, y):     ,  x
198             # function called by self . s wi tcher .     ” control surface callback
199             changeP1(x)
200             self . s wi tcher . changeP2(y)
201
202 app=wx. App ( )
203
204 fxs witch = FxSwitch()
205
206 mainFrame = MyFrame ( s wi tcher=fxs wi tch )
207 mainFrame . Show ( )
208
209 app. MainLoop()
```

scripts/chapter 04/14 final.py