

Affordable Measurement Box for Science

Georges Khaznadar

Wed Aug 17 2016

Contents

1	Booting live USB sticks	3
1.1	We proudly choose an “Insecure” system	3
1.1.1	To sum it up:	4
1.1.2	So to sum up the summary:	4
1.2	Disabling the “Secure Boot” feature	4
1.2.1	To disable Secure Boot:	5
1.3	Now our system is ready, let’s boot with the USB stick	6
2	Why should I use a live USB stick with free software?	6
2.1	Ten reasons why live USB sticks are <i>good</i> :	6
2.2	Ten reasons why free software and open formats will be preferred:	7
2.3	Why would my students like to use copies of my live USB stick?	9
3	First use of the expEYES box	10
3.1	Pickup voltage: where does it come from?	10
3.2	experimenting with a few wires only	13
3.2.1	Fourier transform, page 30 of the manual	13
3.2.2	Using the MIC output	14
3.2.3	A “standalone” experiment, with the DC motor	18
4	Mastering Python	19
4.1	Empowering oneself with Python	21
4.2	A guide to learn Python quickly	21
4.2.1	Level of difficulty	21
4.2.2	Examples of small projects to learn Python	22
4.2.3	Python and expEYES	23
4.2.4	Understanding the first programs for Python+expEYES	24

4.2.5	Proposition of exercises	27
4.3	Python libraries for science	27
4.3.1	<code>numpy</code> : the advantage of using arrays vs. plain Python lists	28
4.4	Python libraries for graphic user interfaces (GUIs)	30
4.4.1	General purpose GUI libraries	30
4.4.2	GUI design programs	30
4.4.3	Our first GUI layout	30
4.4.4	Compiling the layout file to a Python program	31
4.4.5	Our first working program	32
4.4.6	Introducing some “meat” to replace nonsenses of the first program	34
4.4.7	Details of the first modification	34
4.4.8	A few more code, to deal with real life problems and exceptions	35
4.4.9	Exercises: improve our first program	36
5	Writing the user interface for a new experiment	37
5.1	Description of the new experiment	37
5.1.1	Schematics	37
5.1.2	Features wanted for the program	38
5.2	Implementing new features	38
5.3	Coding the new program	40
5.3.1	Designing the user interface	40
5.3.2	Exercise:	40
5.3.3	Minimal structure of the main program	41
5.3.4	Adding methods to toggle the way curves will be plotted	43
5.3.5	Implementing naively the capture of data and the plot- ting routine	43
5.3.6	Less naive implementation for the plotting routine	44
5.3.7	Implementing the callback for the Stop button	45
5.3.8	Implementing the callback for the Save button	46
5.3.9	Implementing the expectation of a rise in the slew rate	46
5.3.10	The fit utility	47
5.3.11	The two help methods	48
5.3.12	Access to the complete source code	50
5.4	Exercises	50

6	Writing documentation for the end user	52
6.1	Key features of an effective collaboration engine	52
6.1.1	Separated edition of neighboring sections	52
6.1.2	Template support	53
6.1.3	Translation support	54
6.1.4	Not WYSIWYG, rather WYSIWYM	54
6.1.5	Free software, open standards	55
6.1.6	From a mediawiki to a book	55
6.2	Exercises: make a small book out of mediawiki pages	56
7	ExpEYES as a common good	57
7.1	Mailing list	57
7.2	Software repositories	57
7.3	Contributing to the software	58
7.3.1	Reporting bugs and feature requests	58
7.3.2	<code>reportbug</code> utility, with an email-enabled system	58
7.3.3	Reports when you are unsure that your system can send e-mails	60
7.3.4	Translating software into your language	60
7.3.5	Translating within a collaborative environment	61
7.4	Exercises	61
8	Conclusion, License	62

1 Booting live USB sticks

1.1 We proudly choose an “Insecure” system

Some laptop manufacturers are applying a new policy proposed by Microsoft: disallow users to boot freely their operating systems, verify that the system loaded during the boot process bears a cryptographic signature approved by Microsoft. This feature is known as “Secure Boot”, as presumably operating systems can be considered as secure, only if they are made by Microsoft company.

So far, Microsoft company does not provide free/libre operating systems, and as we want to distribute live USB sticks which our students will be able to duplicate, understand, and even enhance, Microsoft Windows cannot be used. To have a more precise idea about the problem, let us take a look at the end-user agreement coming with Windows 10:

(excerpt of https://www.microsoft.com/en-us/Useterms/OEM/Windows/10/UseTerms_OEM_Windows_10_English.htm)

1. [...]
2. Installation and Use Rights.
 - (a) License. The software is licensed, not sold.[...]
 - (b) [...]
 - (c) Restrictions. [...] this license does not give you any right to, and you may not: (i) use or virtualize features of the software separately; (ii) publish, copy (other than the permitted backup copy), rent, lease, or lend the software; [...] (vi) reverse engineer, decompile, or disassemble the software, or attempt to do so [...]

1.1.1 To sum it up:

- the software is licensed, not sold: *it can be free as **free beer**, but it is not free as in **freedom**.*
- students (and teachers) are not allowed to publish, neither to distribute copies of the operating system. Even if this may be technically feasible, one can be sued if she does it.
- students (and teachers) will never be allowed to understand how the operating system works. Its source is not published, and attempts to reverse engineer its binaries are forbidden by license.

1.1.2 So to sum up the summary:

AS WE WANT OUR STUDENTS TO BE ABLE TO UNDERSTAND
WHAT THEY USE AND WHAT THEY DO,
WE MUST DEFINITELY OPT FOR INSECURE SYSTEMS.

1.2 Disabling the “Secure Boot” feature

Until year 2015, vendors who wanted to apply the sticker “*Approved by Microsoft*” on their laptop had to enforce “*Secure Boot*” by default, but they used to let a possibility for the users to disable this feature, more or less easily.

More recently, vendors who want to get this sticker are encouraged to remove the possibility of disabling the “*Secure Boot*” feature. Depending on anti-trust law which apply in such or such country, this removal can be considered legal or not. However, one can now find laptops which are difficult to boot without the “Secure Boot” feature.

Here are hints provided by Microsoft Company, about disabling the “Secure Boot” feature (source: <https://msdn.microsoft.com/en-us/windows/hardware/commercialize/manufacture/desktop/disabling-secure-boot>)

1.2.1 To disable Secure Boot:

Before disabling Secure Boot, consider whether it is necessary. From time to time, your manufacturer may update the list of trusted hardware, drivers, and operating systems for your PC. To check for updates, go to Windows Update, or check your manufacturer’s website.

Open the PC BIOS menu. You can often access this menu by pressing a key during the boot-up sequence, such as F1, F2, F12, or Esc.

Or, from Windows, hold the Shift key while selecting Restart. Go to Troubleshoot > Advanced Options: UEFI Firmware Settings. Find the Secure Boot setting, and if possible, set it to Disabled. This option is usually in either the Security tab, the Boot tab, or the Authentication tab.

Save changes and exit. The PC reboots.

Install the graphics card, hardware, or operating system that’s not compatible with Secure Boot.

In some cases, you may need to change other settings in the firmware, such as enabling a Compatibility Support Module (CSM) to support legacy BIOS operating systems. To use a CSM, you may also need to reformat the hard drive using the Master Boot Record (MBR) format, and then reinstall Windows. For more info, see Windows Setup: Installing using the MBR or GPT partition style.

If you’re using Windows 8.1, you may see a watermark on the desktop alerting you that Secure Boot is not configured correctly. Get this update to remove the Secure Boot desktop watermark.

1.3 Now our system is ready, let's boot with the USB stick

When possible, alongside with the “*Secure Boot*” option's tweak, you can choose the order of precedence of bootable devices. If the live USB stick is plugged in, you may reorder boot devices so the USB stick is considered as preferred (coming before the boot on hard disk, and before the boot on network interface). If you could define the USB stick as the preferred boot device, just keep it plugged in while booting.

If you could not define the default boot sequence order, you can choose the boot device on the fly: on most laptops, the key “F8” (or F9, or F12) is dedicated at this purpose. If the key is activated during the early boot process, you are given the choice between a few devices in the so-called “boot menu”. Choose your USB stick, and press “Enter”. Depending on the computer, it can be called *USB disk*, *Removable media*, or be part of a sublist of “*hard drives*”.

When the system can boot from the USB stick, and if the graphic card of the laptop is VESA compatible, you will see a welcome splash screen provided by *SYSLINUX*, and a few options selectable with keyboard's arrow keys. Usually, you just need to wait a few seconds, or you can type “Enter” to select the first entry in advance. Then the boot process begins, and you can see a few messages on the screen, during the hardware check-up. One minute later, the graphic desktop should be usable.

The main menu (applications, settings, etc.) lies in the left bottom corner, you can begin exploring it. When an Ethernet cable is already plugged in, the automatic configuration of network access is attempted, and chances are that you begin with a usable network configuration. There is also a possibility to get network access thanks to WIFI hotspots if any.

2 Why should I use a live USB stick with free software?

2.1 Ten reasons why live USB sticks are *good*:

The following set of reasons is non-exhaustive!

1. I can get an efficient system with a USB stick weighing 8 GB. This is quite affordable. About one half of the storage space is used for the operating system and programs; the second half is available for the “payload”;

2. When I am travelling, or just going from place to place, all I need can fit in my pocket;
3. As far as I want to keep my USB stick *private*, I know that no spy program hosted by the local computer will access my data;
4. Duplicating a USB stick (and all of its features) is a matter of a quarter of an hour;
5. With the KNOPPIX-based live system, I can also duplicate the operating system and the programs without disclosing private data;
6. When used in an educational context, USB live sticks provide an interesting feature: students can replay an exercise, or finish it outside the classroom, with exactly the same software environment which they were using previously;
7. A live USB stick can be used to boot a computer even when it cannot boot on its hard disk, for any reason; it is a valuable rescue system;
8. A live USB stick can survive in a washing machine; a computer cannot;
9. A live USB stick can survive strong accelerations, like falling down-stairs; a computer cannot;
10. **Small is beautiful (see figure 1).**

2.2 Ten reasons why free software and open formats will be preferred:

1. Free software is defined by a list of four elementary freedoms,
 - (a) the freedom to use the software in any case (for example, not only in educational context);
 - (b) the freedom to learn how the software functions. That implies an access to its source code;
 - (c) the freedom to make derivative works based on the software;
 - (d) the freedom to copy and distribute the software, either in its original form or as a derivative work;
2. As, with the definition above, there is no limitation of use (the software *can* be commercial), people can earn their life when working with free software;



Figure 1: Don't keep more in your pocket

3. As the source of the software must be delivered with it, fair interactions can be organized between competitors; every company can access specifications about open formats being used; hence competitors cannot build their wealth upon secrecy; the cost of the software must be justified by its quality;
4. As the source of the software must be transmitted along, the software becomes quite *eternal*, in the sense that you can always hire a skilled person to adapt the software and maintain it if a previous author or vendor fails to do it;
5. Free software can be audited for security issues by specialists, who can know details of its design by studying its source; if a security issue is detected, it can be fixed by the same people;
6. When the source of the software is written in a *human readable* format, one can check that there is no malware included;
7. When the software is used to make science (or to teach science), it is not a “black box”; everyone may watch its internals, and experiments can be readily reproduced worldwide;
8. When there is no constraint on copies or instances of a running software, the management of licenses is way cheaper;
9. The ability for everyone to access the source of the software is a strong incentive for progress and software quality;
10. Most non-free software from the last ten years are now dead; on the contrary, free software, even when it is old, can be revived with some work; the same obsolescence is true with data which were saved in non-open formats, if nobody took the opportunity to convert them to a new format when that was still possible.

2.3 Why would my students like to use copies of my live USB stick?

Here are a few “irrational reasons” why my students like my system based on a live USB stick:

- Thanks to Klaus Knopper’s efforts, every derivative of KNOPPIX, like our USB sticks, comes with an excellent support of accelerated graphic cards, and the desktop is managed by **Compiz**. If you do not know

what **Compiz** provides, you cannot understand why your students are playing with dozens of applications open in separate windows, like jugglers with their balls.

- Creating awesome graphics is a matter of seconds; open Gimp, then File → Create → Logos → (choose a style) → (type your logo's text) → click, *et voilà!*
- They can *cheat when preparing their homework!* For example, the application **wxMaxima** allows them to find immediately the roots of most equations. Of course, when students “*begin to cheat*”, the teacher just needs to propose them more creative assignments: they become able to solve problems which are usually considered out of their reach, at their education level.

And now, here is at least **one** “rational reason” why my students do use my system based on a live USB stick:

- They are assigned **homework** which can precisely be done with the help of that live USB stick!

Of course, the assignments can be done with other software environments, but just gathering the right tools can be a time consuming task: the USB stick concentrates a lot of teaching resources.

3 First use of the expEYES box

3.1 Pickup voltage: where does it come from?

Boot your computer with the live USB stick; bind the expEYES Jr box to the computer with a USB cable, launch the application expEYES Junior (probably a menu item under Education → Science). The window which appears should bear a mention to the detected hardware in its title bar, and feature an oscilloscope screen with one track enabled. In figure 2, one can read “Four Channel CRO+ found expEYES-Junior on `/dev/ttyACM0`”, which means that the application “CRO+” (an enhanced Cathode-Ray Oscilloscope) has detected the box named expEYES-Junior, via the serial port known as `/dev/ttyACM0`.

Take an insulated wire in your hand (do not touch directly the metal), and touch the analogical input A1 with the bare end of the wire. The track on the oscilloscope screen should change a little. Then change the duration of the sampled data set, by dragging the slider “ms/div” to the right (this

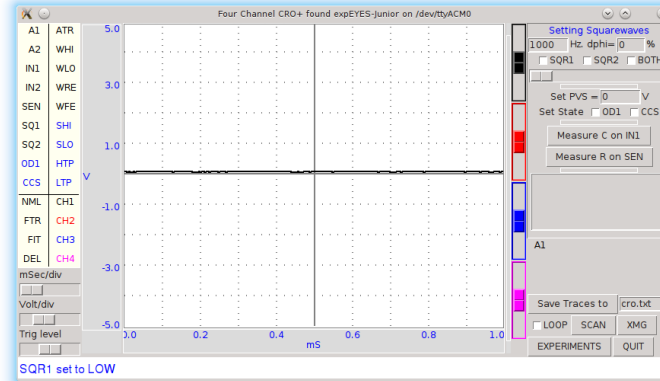


Figure 2: Main window of expEYES-Jr features an oscilloscope screen

slider is on the left of the window, just below the list of channels CH1 ... CH4). When the duration of the sweep is well tuned, you shall see a few wave periods in the oscilloscope's screen.

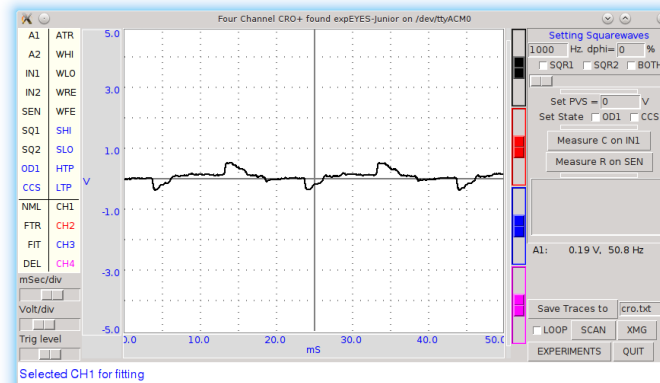


Figure 3: Pickup voltage, time sweep: 5 ms/div

Click the “Save Traces to” button: a file named `cro.txt` (or some other name if you change the default) will be written. You can take a look at this file, with various tools. Try to open the file with the following applications:

1. inside a Terminal, type the command `cat cro.txt` (and type *Enter*): you will be provided many lines of text, which you can scroll up and

down with the vertical slider of the terminal. The command `cat` is a very simple command: it opens one or more files given as arguments, and concatenates their contents to the standard output, which is the terminal's display in that case.

2. inside the same terminal, type the command `less cro.txt` (and type *Enter*): you will be provided the same set of lines, but you are using a so-called “pager” (the command `less` calls the default pager of the system). You can access previous and following lines by using vertical arrow keys, or Page-Up Page-Down keys. Type “Q” to stop the pager program and come back to the terminal's prompt.
3. launch the application Qtiplo (probably available under the menu Education → Science). Then, import data from the file `cro.txt`: Window's Menu → File → Import → import an ASCII file, or quicker with the graphic button which does the same (an icon with “123” written above), or else with the keyboard shortcut Ctrl+K. Choose the right file, import it. If numbers are not well imported, you may want to change some import options, like the number's input format. The values from the file `cro.txt` will be displayed in a data grid. You can select both columns with the mouse, then plot the data easily (either by calling a contextual menu by a right click, or by using the “Plot” window's menu). The application Qtiplo has many features to analyze data series, which deserve a detailed study.
4. you can also import the data with LibreOffice's spreadsheet program, known as `Calc`. When data are there, other features of the spreadsheet can be enjoyed.

While maintaining the right sampling settings, use the left mouse button to drag the symbol “CH1” to the symbol “FIT” in the left part of the screen. The application will begin to fit the sampled data with a template function based on a sinus. The amplitude and the frequency should appear in “*real time*” near the oscilloscope's screen.

In order to reply to the title's question, while maintaining the wire connected to *A1*, one hand on the insulating part of the wire, move your body, or just a member, nearer or further from areas where power lines are visible or hidden. If the “FIT” feature is enabled, you may notice that the amplitude depends strongly on the proximity between your body and power lines, while the frequency remains quite unchanged. One can read “A1: 0.19 V, 50.8 Hz” in the middle of the right part of the window, in figure 3.

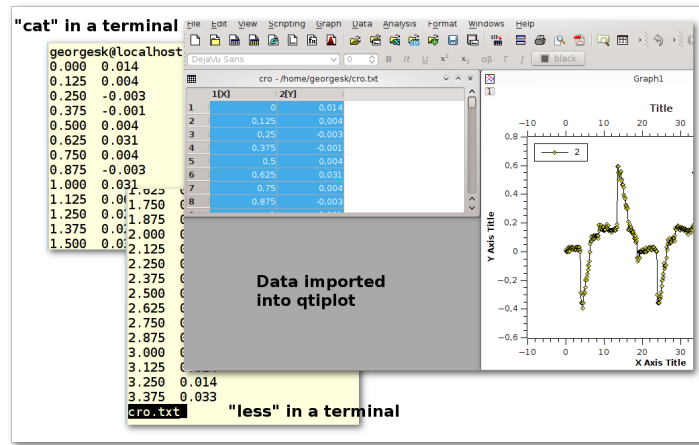


Figure 4: The same data, accessed by “cat”, “less” and “qtplot”

Another question which is worth a quick search for documentation by students is “*why precisely 50 Hz?*”. Wikipedia’s web site gives valuable information about the history of that frequency standard.

3.2 experimenting with a few wires only

ExpEYES-Junior is a measurement box, but not only that: it is enhanced by numerous generators. Additionally, the program **expeyes-junior** comes with some powerful features, like one-click Fourier transform.

3.2.1 Fourier transform, page 30 of the manual

For a first try, let us open the *User Manual*, page 30.

It is a simple manipulation about Fourier transforms, for a sinusoidal and for a square wave.

The manual tells to connect two wires, one from *SINE* to *A1*, the second from *SQR1* to *A2*. You can also use four short wires with crocodile clips, and do the same by joining clips accordingly.

When the box is bound to the computer, and the wires are in place, please launch the program **expeyes-junior**, set a frequency $f = 1500\text{ Hz}$ for *SQR1* and enable the second track *CH2*. Figure 5 shows two oscilloscope tracks, we can see that *SQR1* is enabled, with a frequency of 150 Hz.

The symbols *A1* and *A2* in the left column of the window have been dragged with the mouse and dropped on the symbols *CH1* and *CH2* respec-

tively.

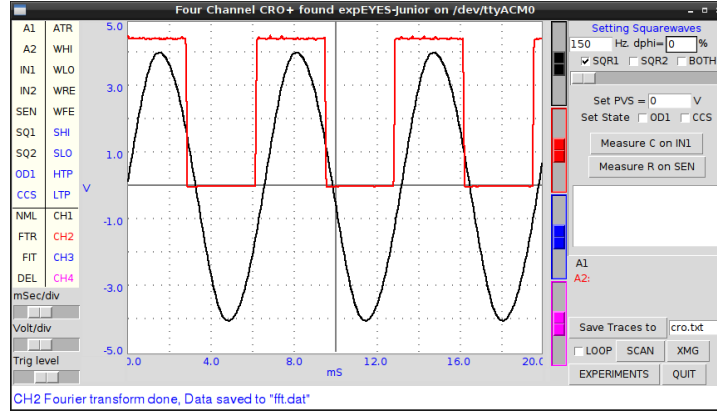


Figure 5: Two tracks of the oscilloscope are used, the frequency of both signals are quite the same

In order to get a Fourier transform of the signals, the manipulation is very simple: when you drag the symbol CH1 on the neighboring symbol FTR, you get a new window like figure 6, and when you drag CH2 on FTR, you get another window like figure 7. The Fourier transform is done quickly by the application. You can notice more peaks in the FFT plot of the square wave. The first peak is at frequency zero (it is due to the non-null DC component), and other peaks are at frequencies which build an arithmetic progression (450, 750, 1050, 1350 Hz). You may notice that *even* harmonics of the signal do not appear.

3.2.2 Using the MIC output

Connect a wire (or two wires bound by crocodile clips) between the *MIC* socket, which is the output of an amplified microphone, and *A1*, the first analog input. Then, you must provide some “musical” sound near the microphone, which is placed at the left bottom of the expEYES box. For example, use some whistle, or sing a single tune during a few seconds. Adjust the time sweep in order to have a few waves in the oscilloscope’s display. Begin to drag the symbol *CH1* above the symbol *FTR*, and drop it when the oscilloscope’s track is well shaped. This records data in two files: *cro.dat*, which contains original data re-sampled with a time step ensuring a nice Fourier transform, and *fft.dat*, which is the Fourier transform of the same data, to give a spectrum (amplitude *vs.* frequency).

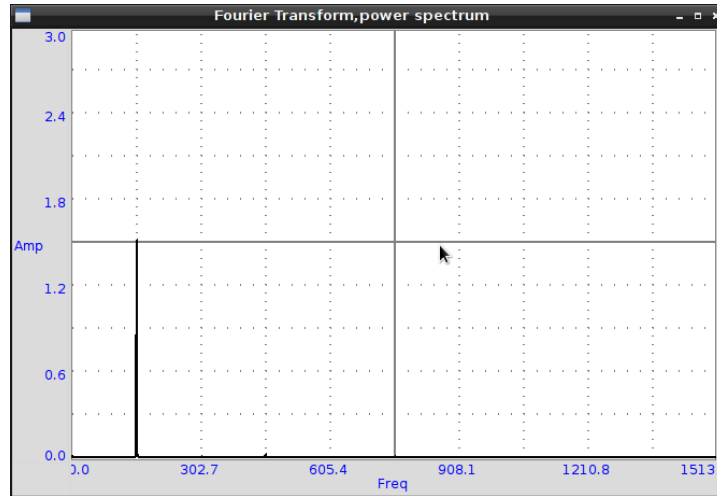


Figure 6: Fourier transform of the SINE wave

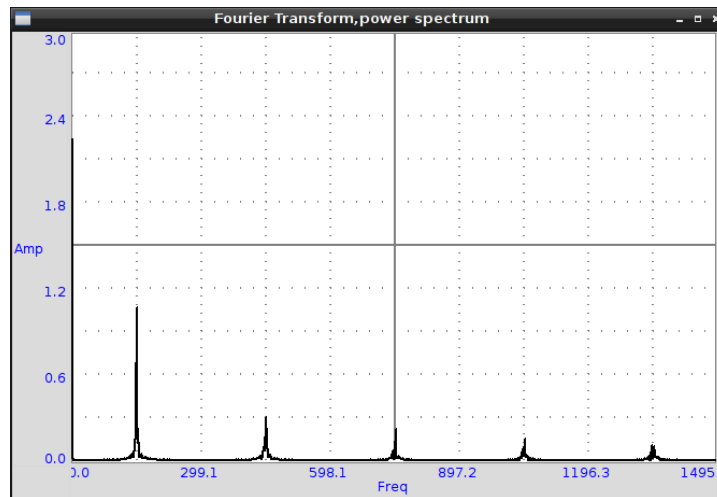


Figure 7: Fourier transform of SQR1's signal at 150 Hz

The figures 8 to 11 show the data recorded, first with a voiced “AAA”, then with a mouth-whistled still tune of a higher frequency.

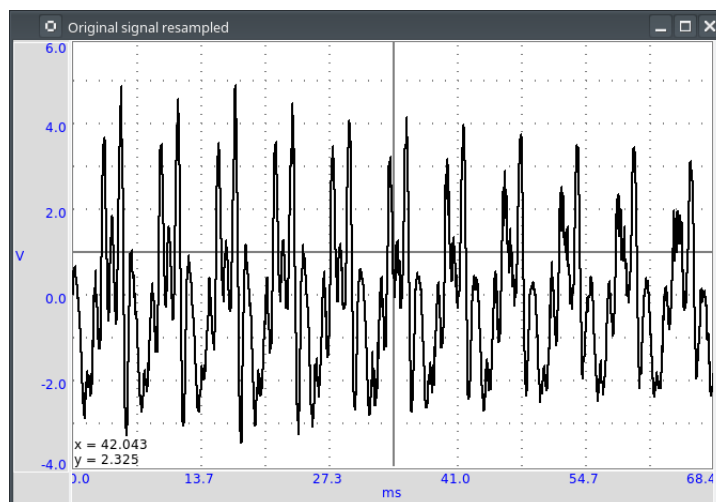


Figure 8: A voiced “AAA”: the original signal

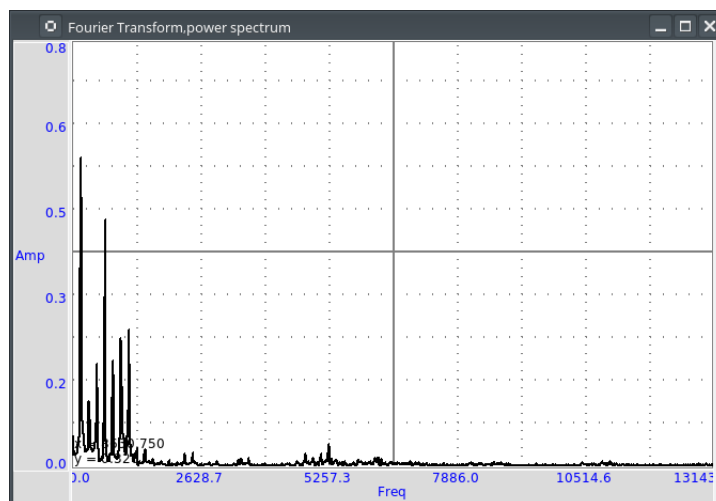


Figure 9: A voiced “AAA”: Fourier transform

Further is another analysis of a mouth-whistled tune (figure 12), done with the file `cro.dat` which was imported (as ASCII data) in the application `qtplot`. You can notice that the abscissa for frequency should be kHz (not

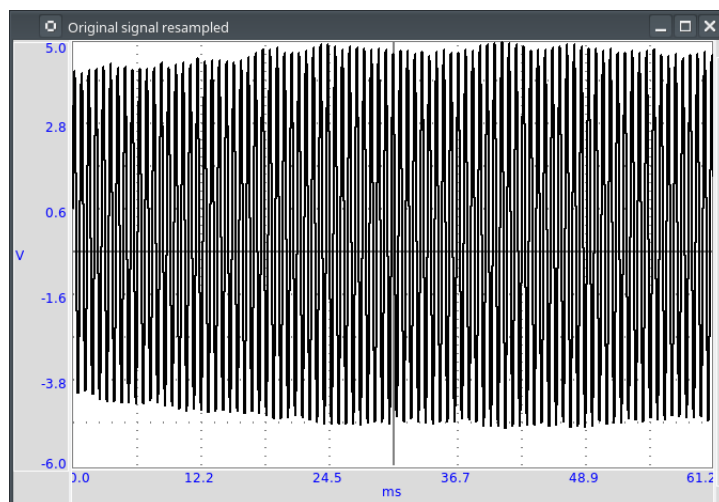


Figure 10: Mouth-whistled still tune: the original signal

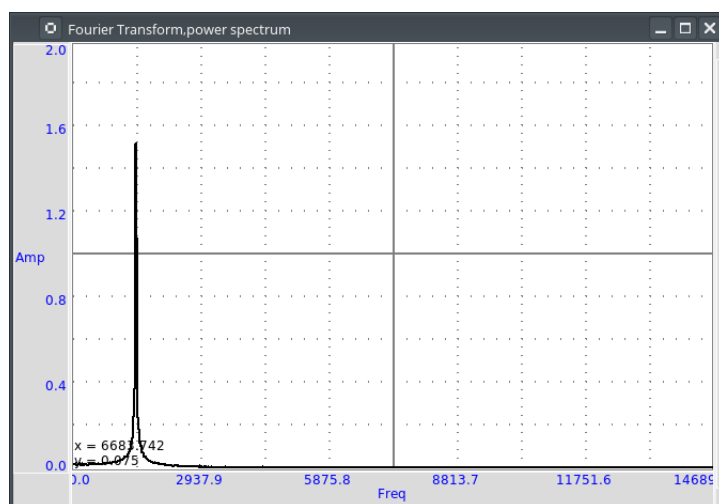


Figure 11: Mouth-whistled still tune: Fourier transform

Hz), because the unit for time exported from `expeyes-junior` is *millisecond*, not *second*.

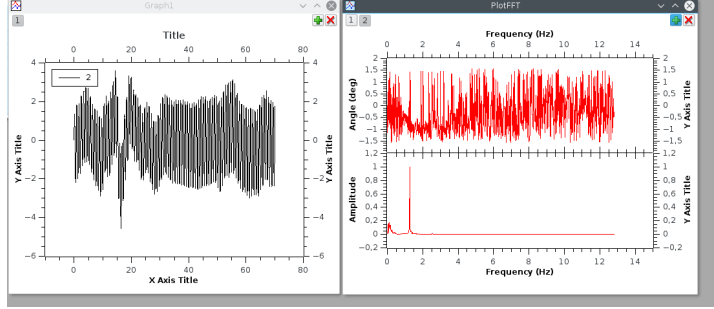


Figure 12: Another mouth-whistled still tune, analyzed by qtiplot.

3.2.3 A “standalone” experiment, with the DC motor

The expEYES-Junior kit comes with a DC motor (this model is ordinarily used to drive a CD-ROM); this DC motor can act as a low voltage motor, but it is also an efficient tachometer, that is, an angular velocity probe: its output voltage is proportional to its angular velocity, and as it has little internal friction, it can be used to measure the movement of a pendulum. When the main application `expeyes-junior` is running, click on the “*EXPERIMENTS*” button: a submenu appears, where you must activate the entry named “*Pendulum Waveform*”.

The window of the main application remains in place, but it loses the control of the experiment box, which is given to a specialized application. To enjoy the secondary application, you must make the following connections:

1. from inverting amplifier’s *OUT* plug to analog entry *A1*, with one wire;
2. the motor’s wires are connected between the ground (*GND*), and the input (*IN*) of the inverting amplifier.

Then, you can check that the second application begins to record voltage data as soon as the motor has enough angular velocity. You can stop the record at any time with the “*STOP*” button.

Put the motor’s axis horizontal, above the edge of a table, and grasp firmly the motor’s body put on the table top. Fit the motor’s axis to a pendulum; there are many ways to do it, all are correct, as long as most of

the mass of the pendulum is far enough from the motor's axis, and that the pendulum remains tightly fixed to the motor's axis during oscillations.

Here are two easy and cheap solutions used by the author:

- either a steel pendulum coupled to the motor's axis with a small and strong magnet,
- or a wood stick fastened to the motor's axis with a screw for wood of diameter 2 mm , with some relatively heavy clip fastened at the other end of the stick, see figure 14.

While the motor is firmly grasped, slowly put the pendulum far from its equilibrium position, and let it swing freely. The angular velocity probed by the motor goes quickly above the threshold which triggers the record's begin. The record's duration is *15 seconds* by default (but you can change it). When the record is finished, you should click on two buttons, first "*SAVE To*", later "*FIT*". So, you get a file with the data recorded, for further analysis, and the *FIT method* gives you two interesting coefficients: the oscillation's pseudo-frequency, and its damping coefficient; see figure 13.

The last figure of this section (figure 15) shows the data from the file `pendulum.dat` reworked with `qtiplot`'s analysis tools (Analysis \rightarrow Fit Wizard ... *define a user fit*). You can see that the chosen formula is $A1 * \exp(-x/A2) * \sin(2 * \pi * A3 * x + A4)$; so, it is a damped sinusoidal function, with damping coefficient $A2$, frequency $A3$, and phase $A4$; $A1$ is a multiplying factor. To perform the fit, their values were initialized like: $A1 = 1.5$ (the order of magnitude of the initial amplitude), $A2 = 10$, as the movement is significantly damped within 10 seconds, $A3 = 1$ since the pseudo-frequency is near 1 Hz , and $A4 = 1.5$ (which is near $\pi/2$). The fitting algorithm of `qtiplot` yields a damping factor $A2 = 1.2\text{ s}$ and a frequency $A3 = 1.069\text{ Hz}$, with an excellent correlation factor: $R^2 = 0.993$. The precision is greater for the frequency than for the damping factor.

4 Mastering Python

Python is an interpreted computer language created by *Guido van Rossum* at *Centrum Wiskunde & Informatica (CWI)* in the Netherlands, fifteen years ago approximately. Currently this language is considered as excellent for dealing with scientific and experimental data, and most important software libraries which can be used in that area have been linked to Python and are providing well-documented Programming Interfaces.

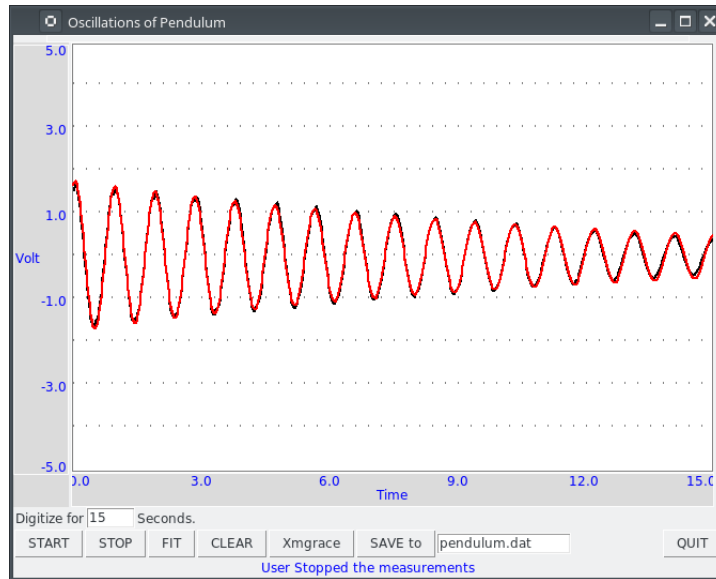


Figure 13: Record of a pendulum's angular velocity

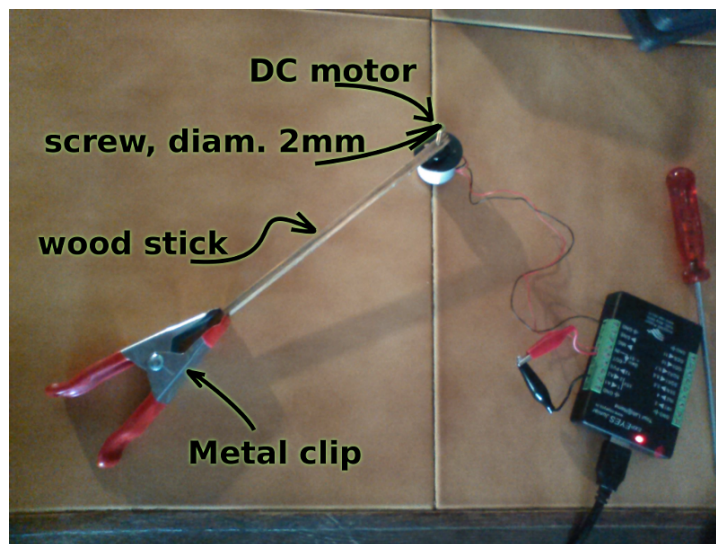


Figure 14: Pendulum made with a wood stick, a screw and a clip.

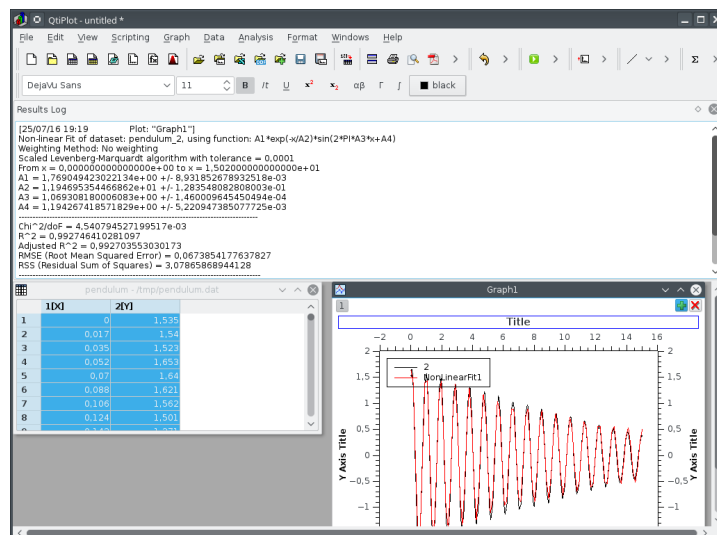


Figure 15: Reworking the data with qtiplot.

4.1 Empowering oneself with Python

Archimedes said:

Give me a place to stand and with a lever I will move the whole world.

He was probably thinking about Python ;).

Computers come now with unprecedented arithmetic power, and there are clever programs which make it usable for calculus also. Of course, students must know how to make operations by themselves, but when thousands of calculations are necessary, controlling a computer is *the solution*. Python is one of the shortest ways to control this power.

Incidentally, the way Python programs must be written makes them often quite readable by non-aware people. I could discuss with a teacher of philosophy who showed me a source program and pointed one precise part, saying “there is an error, right there!”.

4.2 A guide to learn Python quickly

4.2.1 Level of difficulty

Learning to walk with one’s own legs is a non-trivial task, most humans achieve it in their early youth. Driving a helicopter is also a non-trivial task,

but few humans achieve it, even if it might give them huge advantages. There are reasons why few persons can learn to drive a helicopter: the learning environment is very expensive, and also somewhat risky: a beginner can destroy a helicopter and herself in the same run, with some probability.

Learning arithmetic operations is also a tough task, which most persons achieve in developed countries. Controlling a computer, compared to basic arithmetic, is more than like driving a helicopter, compared to walking. However, unlike the helicopter example, it is much safer: Python is a free-libre program, you can get it gratuitously; accidentally breaking a computer while learning computer science with Python is most unlikely.

In a few words, the best guide to learn Python quickly is your own curiosity, as long as you choose a project which makes sense. Good tutorials exist in many places, here is the tutorial of the Python Foundation: <https://docs.python.org/3/tutorial/index.html>. Begin with your own project, and schedule reasonable milestones. At every stage of the project, you can access many examples, and valuable explanations if you use research engines in Internet.

Today, there are two families in the Python language, known as Python2 and Python3; the fork between both families has begun a few years ago. If you are beginning, you should definitely learn Python3, as it is the only branch which will be maintained and developed in the future, even if many efforts are still deployed to help the large community of people who are still using Python2.

4.2.2 Examples of small projects to learn Python

1. Repeat writing the same line of text many times
2. Write lines of text made of some character, for example a star (*), which will build some interesting shape when watched from some distance: a ball, a robot, a toy, a pet, etc.
3. Produce automatically arithmetic tables: for addition, multiplication, etc. and why not, logarithms too?
4. Create some interactive game: the computer asks a question, the player replies, the further exchanges depend on the replies, etc. For instance, the computer chooses a random number within a range, and the player has a limited number of questions left to know which it is ...
5. Draw an elephant, which will cross the screen, coming from the left and going to the right (or the opposite)

6. Create a structure which can be used to manage your collection of music records, and two different routines to display collection's items, with more or less details
7. Create an interactive program which allows one to query your collection's items, based on search criteria (author, some words from the title, music style, etc.)
8. Write a program which quickly solves equations of degree one, two ... three, four (if you dare to go that far)
9. Write a program able to say how many different colors there are in a picture
10. Write a program which automatically detects a human face in a given picture

Propositions of solutions for this set of exercises are published in the “repository”.

4.2.3 Python and expEYES

Your computer does not work in “real time”: that means that when you begin some task, you cannot require with certitude a precise duration for that task to complete. For example, if you want your computer to measure a thousand times the voltage of a precise pin during one second, you cannot be sure that it will perform this operation uninterrupted.

This is why the physical measurements are done inside expEYES box with a separate micro-program designed to work in *real time*. Then, your computer can ask the box to make the thousand measurements, they will be done *uninterrupted*. Later, the results will be given back to your computer, possibly with some interruptions, but you are sure that the data were captured on a regular time basis. As the computer, despite its many interruptions, works very very fast, you experience a rich feedback, not unlike the feeling you can have with an oscilloscope. However, because of its calculation capability, your computer can go much further than an oscilloscope, and this is why Python is used, to enjoy the many libraries able to crunch numbers for us.

Now, let us begin with Python+expEYES: let us open the application **geany** (a good development environment for programmers... Windows users will probably use **Notepad++**), and type the few lines presented in figure 16. Then, save them with the filename **p1.py**. As soon as the name of the file

is defined, **geany** will use colors and styles to make the syntax of the source file more evident. For example, keywords like **import**, **print**, are displayed in a particular fashion.

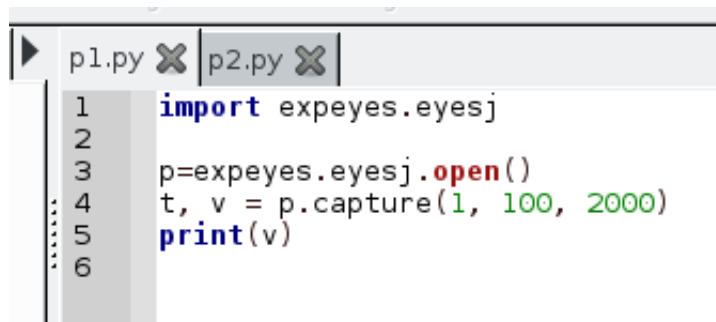


Figure 16: Our first program with Python+expEYES ever.

When the file is saved, press the “F5” key (it is a shortcut to launch the program under edition). You will get a heap of numbers written in a *Terminal*, and you will be prompted to press the “Enter” key. When you press it, the *Terminal* is closed. The series of numbers are a sequence of voltages measured very fast on the analog input A1. If you want more variety inside this sequence of numbers, you can connect a wire to this input and act exactly as for the previous experiment "Pickup voltage". The number sequence can be impressive, but it not as expressive as a good plot. In order to make a plot, you need three more lines in the program. Save the previous program with a new file name, like **p2.py**. So, the first program still exists and can be used as a begin point for other variations. Then add three more lines, as showed in figure 17. Save the program and type “F5”. If there is some “pickup voltage”, you should get a plot like in figure 18.

You can notice that the plot comes in a separate window which has its own close button, and that you will have to close the *Terminal* too after the end of the program.

4.2.4 Understanding the first programs for Python+expEYES

Let us comment the few lines in the program **p1.py**:

```
import expeyes.eyesj: this declares that the library which drives expEYES-  
Junior will be used; the library's name is expeyes.eyesj, if we want a  
shorter alias, it is possible to declare it, more about that later ...
```



```
p1.py p2.py
1 import expeyes.eyesj as ej
2 from pylab import *
3
4 p=ej.open()
5 t, v = p.capture(1, 100, 2000)
6
7 plot(t,v)
8 show()
9
```

Figure 17: The second program with Python+expEYES.

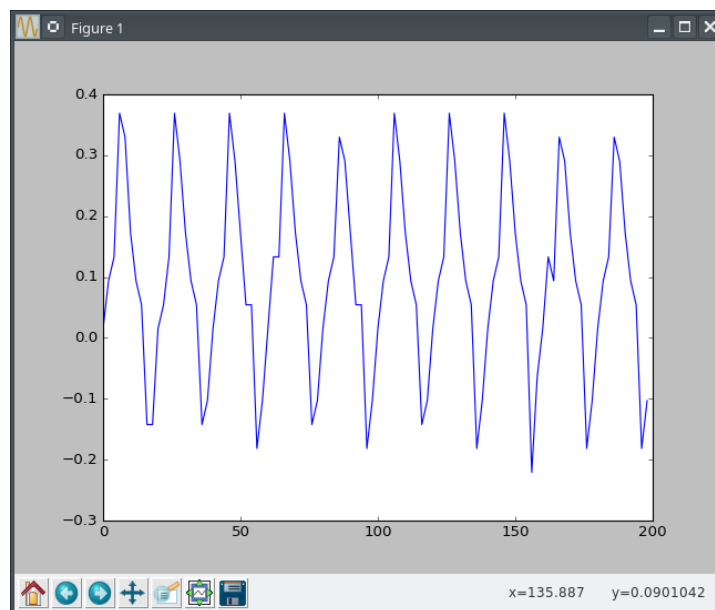


Figure 18: The plot given by the second program

p=expeyes.eyesj.open(): this line calls the function `open()` from the library which drives expEYES-Junior, and the resulting object is stored into the *variable* `p`; hence, `p` contains the initialized driver to deal with the measurement box.

t, v=p.capture(1,100,2000): this line calls the method `capture()` from the driver, with three parameters:

1. 1 is the number of the input channel; 1 stands for channel *A1*
2. 100 is the number of samples to collect: a hundred of measurements are requested
3. 2000 is the duration between two consecutive measurements: measurements will be done every $2000\text{ ns} = 2\text{ }\mu\text{s}$; so the total duration of the “experiment” is requested to be $200\text{ }\mu\text{s}$. The results are assigned to *variables* `t` and `v`, `t` is the list of timestamps, `v` is the list of measured voltages.

print(v): this line asks to print the hundred values measured in the *standard output* of the program, which is the *Terminal*, there.

Now let us comment the new lines in the program `p2.py`:

import expeyes.eyesj as ej: exactly the same as in `p1.py`, but the alias `ej` will be used to name the library.

from pylab import *: this line means that all objects and functions of the library `pylab` will be available directly in the program. Such an instruction may cause collisions if two libraries provide a same name for different functions. Here there will be no problem because the program is very short. The functions of interest are `plot()` and `show()`.

p=ej.open(): exactly the same as in `p1.py`, you can notice that the alias `ej` is used there.

plot(t,v): this is the call to a function provided by the library `pylab`, it builds a simple plot with the series of values `t` as abscissa and the series of values `v` as ordinate.

show(): this call displays the graphic window which contains the plot, and allow users to interact with it. This window “lives” independently of the main program.

4.2.5 Proposition of exercises

As you can see, the plot in figure 17 misses details, and some features of the expected waveform are not as precisely rendered as they might be.

You can make a series of exercises, in order to:

- make more measurements, in the same total time
- make a longer measurement, with more samples but same laps of time

You can also try to find the limits of the measurement box:

- which is the shortest duration between two consecutive measurements?
- how many measurements can be stored in the box for a one-shot measurement series?
- how many different inputs has the box, which numbers are they assigned?
- etc.

Propositions of solutions for this set of exercises are published in the “repository”.

4.3 Python libraries for science

As we already know, Python exists for fifteen years approximately, and most important free and open-source libraries have been made available for Python, thanks to a versatile toolset which allows one to link C and C++ libraries to Python language.

You can access a good review about Python Libraries for Science at <https://wiki.python.org/moin/NumericAndScientific>. Here are the two first entries of this webpage, as captured in August 2016:

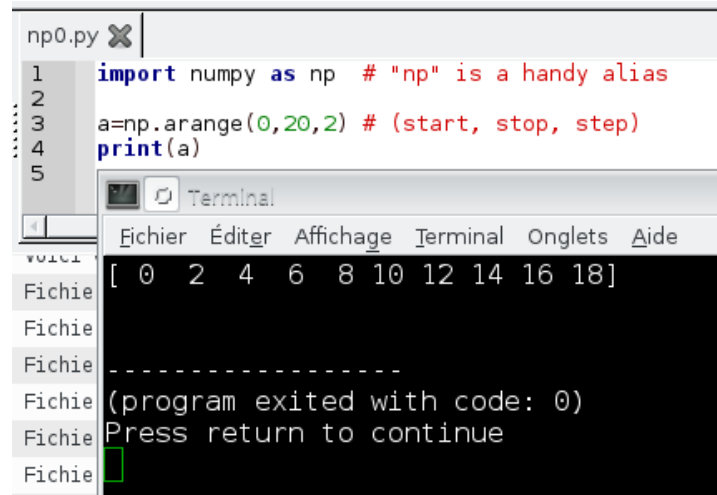
NumPy: <http://www.numpy.org/> – Numerical Python adds a fast, compact, multidimensional array facility to Python. [...]

SciPy: <http://www.scipy.org/> – SciPy is an open source library of scientific tools for Python. SciPy supplements the popular NumPy module, gathering a variety of high level science and engineering modules together as a single package. SciPy includes modules for linear algebra, optimization, integration, special functions, signal and image processing, statistics, genetic algorithms, ODE solvers, and others.

4.3.1 numpy: the advantage of using arrays vs. plain Python lists

When one uses plain Python, she can manipulate series of numbers as so-called *lists*: for example the 10 first even integers are this Python list: `[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]`; such a list can have a memory footprint bigger than ten times the place necessary to store an integer, because Python lists can contain any type of element (for example, a list can contain another list), and such a flexibility requires more memory space and sometimes slower algorithms to take in account all the possibilities.

The module `numpy` allows you to declare or to construct numeric arrays, which contain only numbers, and which can be computed efficiently. The figure 19 shows the code to get the same numbers as above in an array.



```
np0.py X
1 import numpy as np # "np" is a handy alias
2
3 a=np.arange(0,20,2) # (start, stop, step)
4 print(a)
5
```

Terminal

Fichier Éditer Affichage Terminal Onglets Aide

```
[ 0  2  4  6  8 10 12 14 16 18]

-----
(program exited with code: 0)
Press return to continue
█
```

Figure 19: ten first even numbers, with numpy: source code, and output

The function `arange` allows us to build very easily long arithmetic series. For example `arange(0, 2, 1e-3)` yields an array of two thousand numbers.

Numpy provides also mathematical functions and operators which work directly on arrays; here are two example programs, in figures 20 and 21.

Both compute $y=2*x$ and $z=\sin(x)$ for any element of an arithmetic series, we can notice that `numpy` allows us to use a more clear programming style!

```
np1.py ✕  
1  import math  
2  
3  x=range(0,20,2)  
4  y=[]  
5  z=[]  
6  for element in x:  
7      y.append(2*element)  
8      z.append(math.sin(element))  
9
```

Figure 20: some math with number series, without numpy

```
py ✕  
import numpy as np  
  
x=arange(0,20,2)  
y=2*x  
z=np.sin(x)  
|
```

Figure 21: some math with number series, and numpy

4.4 Python libraries for graphic user interfaces (GUIs)

Some Python modules used for science can provide good tools for user interaction: for example, the module `pylab` provides two functions, `plot()` and `show()` which are very handy to get a simple plot in an interactive window: users can zoom in some part of the plot, and save it in a file, etc.

4.4.1 General purpose GUI libraries

However user interfaces must often contain some controls to let the user trigger a physic interaction, or to launch some process on captured data. So, general purpose GUI (Graphic User Interface) modules are useful. A good GUI library allows one to internationalize easily a program, making it usable by wide communities. Additionally, good GUI libraries inherit knowledge about human-machine interaction, and the author of programs can focus her attention about things which really matter for the end user: creating an *intuitive interface*, *i.e.* reusing popular widgets with well-known behaviors, defining contextual help phrases, making the current status well understandable, etc.

One good practice is to develop separately the user interface's layout and the program's processes which do the “backoffice work”.

4.4.2 GUI design programs

Some popular GUI libraries come with a specialized application to design the GUI layout. For Gnome Toolkit libraries (Gtk), it is known as `glade`; we shall make some exercises below with `designer`, which comes with Qt libraries. Both `Gtk` and `Qt` libraries work well with Python, and they can target a wide list of platforms, ranging from desktop and laptop computers with various operating systems, to mobile devices.

4.4.3 Our first GUI layout

Let us create our first GUI layout with `designer`, which will feature:

1. one area to plot scientific data
2. one button to launch a specific process
3. one button to quit the application.

To begin with, the application `designer` must be launched without any option. So it will display a menu and we shall choose the `Main Window`

item. An empty Main Window prototype appears in the center of the screen, collections of widgets are proposed in a column at the left side.

Scroll down this column, select the widget known as `QwtPlot` which is part of the “*Qwt widgets*” family, and drag its icon to the top of the Main Window prototype.

Then scroll up the widget column, select a `Horizontal layout` widget, and drag it onto the Main Window, drop it below the `QwtPlot` footprint. Then, you must drag and drop two `Push Button` widgets onto the `Horizontal Layout`. The first one is easily dropped. In order to get a usable structure, the second one must be dropped when a dark blue line appears on the west or on the east border of the `Horizontal Layout`, meaning that the second button will be organized horizontally either on the right or on the left of the previous button. If you want the buttons to be packed on the right, you can drag and drop a `Horizontal Spacer` widget at the east border of the `Horizontal layout`. Then, click on some free area of the Main Window to select it, and click on the “Vertical Layout” tool button, in the upper part of the window, just below the menus. This will organize all of the Main Window.

Finally, save your work (File → Save) with some filename like `p1.ui`

It is better to add more attributes to the push buttons. Click on the left push button to select it, then browse its attributes in the middle of the right column of `designer`’s window. There, you should modify the button’s name to “measureButton” (`QObject` → `objectName`) and its label to “Measure” (`QAbstractButton` → `text`). Select the right push button, and make similar modifications: its name should be “closeButton” and its label should be “Close”.

Save you work again. It is finished, you can close the application `designer`.

The suffix `.ui` of the saved file means that this a file to describe the layout of the GUI in an open format. More than an application is able to read and to interpret this format efficiently and to compose a sensible user interface.

4.4.4 Compiling the layout file to a Python program

1. Open a Terminal, ensure that the default directory contains the recently made `p1.ui` file, type the command line `pyuic4 p1.ui -o UI_p1.py`, and hit the Enter key. This will create a new file in Python language with the name `UI_p1.py`.
2. Open that last file with a source editor, go to the end, and make a small

change to the last line: instead of `from qwt_plot import QwtPlot`, this line should contain: `from qwt import QwtPlot`, to comply with the name of the module inside a Debian distribution (I assume that we are working with the USB live stick). This second operation should not be necessary, but a small bug in the Debian package of `pyuic4`, the converter from `designer's .ui` format to Python source, makes this little modification mandatory (for now).

4.4.5 Our first working program

Here is our first program which will rely upon the just created user interface. The series of snapshots (taken from `geany's` buffers) come with some additional comments in the captions.

```

1  #!/usr/bin/python3
2
3  import sys
4  from PyQt4 import QtGui, QtCore
5  from UI_p1 import Ui_MainWindow
6  from qwt import QwtPlotCurve
7  import numpy as np
8
9  class MyWindow(QtGui.QMainWindow):
10     """
11     This class implements a derivative of
12     PyQt4.QtGui.QMainWindow, a complete application
13     window, which can feature menus, submenus,
14     status bar, etc. In this example, it uses
15     few of those features.
16     """
17

```

Figure 22: Program's header with inclusion of modules and definition of our custom Main Window

```

17
18 def __init__(self, parent=None):
19     """
20     Constructor: creates an instance of MyWindow
21     """
22     #####
23     # Necessary actions, which must be done #
24     # for any project                        #
25     #####
26     # first, calling the ancestor's creator
27     QtGui.QMainWindow.__init__(self, parent)
28     # get the User Interface from the module UI_p1
29     self.ui=Ui_MainWindow()
30     # initialize the user interface
31     self.ui.setupUi(self)
32     #####

```

Figure 23: The constructor of our custom window: mandatory part


```

31         self.ui.setupUi(self)
32         #####
33         # Custom actions, which can be written #
34         # in other ways for other projects.    #
35         #####
36         # aliases for some parts of the user interface
37         self.plotWidget = self.ui.qwtPlot
38         self.measureButton = self.ui.measureButton
39         self.closeButton = self.ui.closeButton
40         # connect methods to buttons' click signals
41         self.measureButton.clicked.connect(self.measure)
42         self.closeButton.clicked.connect(self.close)
43         # initialize an empty curve for the plot widget
44         self.curve = QwtPlotCurve()
45         self.curve.attach(self.plotWidget)
46         return

```

Figure 24: The constructor of our custom window: custom part

```

47
48     def measure(self):
49         """
50         This is a custom method to connect to the
51         button for measurements.
52         There is no need for another custom method,
53         since the method "close" is already inherited
54         from the ancestor class.
55         """
56         # create data for a curve with some fixed
57         # and some random features
58         import random
59         x=np.arange(0,8,1e-2)    # abscissa: [0, 0.01, 0.02, ... 7.99]
60         r=random.random()
61         y=np.sin(x)+r*np.sin(3*x) # calculated ordinate
62         # feed new data into the curve
63         self.curve.setData(x,y,len(x))
64         # change the title of the plot on the fly
65         self.plotWidget.setTitle("sin(x) + {} sin(3x)".format(r))
66         # display the result
67         self.plotWidget.replot()
68         return
69

```

Figure 25: our custom window's "measure" method

```

69
70     if __name__ == "__main__":
71         app=QtGui.QApplication(sys.argv)
72         window=MyWindow()
73         window.show()
74         sys.exit(app.exec_())
75

```

Figure 26: footer of the program: what to do if the file is called as a main program

You cannot copy and paste the screenshots, however the complete program `p1.py` can be downloaded from our repository at github.com.

Download the file (`p1.py`) and open it in a good editor to check its content. It should work easily. For example, you can launch it as a main program by typing `python3 p1.py` and hitting the *Enter* key, provided the default directory of the Terminal contains `p1.py`.

Don't be afraid by this program! Obviously, seventy lines in a language which you do not yet master can be an issue, but they are less an issue if you consider that you can freely copy this program source, and that you become really productive, just by modifying small details inside its structure.

For example, take a look at lines 56-68: they are there like a *placeholder*, for some useful program which you want to write. Plotting a function with random values is quite a nonsense, but it will become far from a nonsense if you replace those lines by a program to plot captured physical measurements.

4.4.6 Introducing some “meat” to replace nonsenses of the first program

The two subsequent programs will show you how to make some sense with the previous program, by wisely replacing some of its structure.

You can download here the second program, named `p2.py`. The modified parts are:

1. the program's header, to include one module for expEYES
2. the `__init__` constructor, which initializes the expEYES Junior box
3. the `measure` method, which captures physical data thanks to the box, and plots them

4.4.7 Details of the first modification

modifications of the header part: The line `import expeyes.eyesj as ej` allows us to control the expEYES Junior measurement box from Python.

modifications of the constructor: The single line `self.p = ej.open()` attaches expEYES' driver object to the main window.

modifications of the measure method: Only two lines are used: `t,v = self.p.capture(1,1000,200)` to request a thousand voltage measurements, and `self.curve.setData(t,v,len(t))` to plot them.

The conclusion is, that **four lines of code** are sufficient to turn a general-purpose GUI program into a usable scientific tool.

4.4.8 A few more code, to deal with real life problems and exceptions

The program `p2.py` works very well, with one exception: when no expEYES Junior box is connected to the computer, nothing happens as expected. If the program was launched from a command line in a Terminal, you may notice error messages appearing there. But if the program is launched by some other method, chances are that it will not work, and the reasons why it doesn't, are not obvious for the average user.

The designer of a program which touches physical realities must always keep a thought about what can go wrong; the most usual issue is a false connection, or no connection at all. Here we have an example: if the expEYES Junior box is not connected, the application should provide an a strong enough feedback for making the end user aware of the probable failure's reason.

Fortunately, Python offers a nice structure to deal with exceptions, the construct `try: {some code which might trigger a failure} except: {something to do if the failure was triggered}`. You must write two blocks of code, the first one is governed by the keyword `try:`, and it must raise an error (in the Python sense) if anything goes wrong; the second one is governed by the keyword `except:` and contain instructions to process if an error was raised in the previous block.

Here you can download a third program, which can deal with mistaken connection to expEYES Junior (or any error reason which can fool the detection of the hardware)

The differences between programs `p2.py` and `p3.py` are few:

In the `try: clause`, the line `self.p = ej.open()` is now embedded in the `try: block`. A condition is tested just after this line: `assert(self.p.fd)` is an affirmation that the driver has a working file descriptor; unless this descriptor exists, a Python error is raised (you would see it if you write the same program line without the `try: block`); if this descriptor exists, the next lines change the title of the Main Window. In most cases, this title will become “**expEYES Junior found on port /dev/ttyACMO**”

In the `except: clause`, there are two lines of code, to manage the exception. The first one assigns the Main Window's title to “**ERROR:**

expEYES Junior NOT FOUND!”. The second one disables the button “Measure”, since this button should not be used to make measurements.

4.4.9 Exercises: improve our first program

Here are some exercise propositions, which can make you more fluent with GUI programming. Some are modifications touching just the user interface’s design, others touching the way captures are done, or implementing additional features.

1. Change the user interface file, replace the label “Measure” by some other label, like “Launch a capture”, for example
2. Add a button to save data into a file named “data.csv”; verify that this file can be readily opened by other programs like LibreOffice Calc.
3. Add a text input allowing end users to choose another file name
4. Change the capture instruction, in order to get a thousand voltage values, separated by time gaps of two milliseconds; the total capture time should be two seconds
5. Two seconds is quite a long time for our impatient students. Try to implement some mean to provide a feedback message stating “Measuring voltage... please wait” from the click’s time until the plot’s update.
6. Add a text input to let users request a precise number of voltage samples, so they can ask for as many samples as they want inside a reasonable range.
7. Add text inputs to let users request other input channels than “A1”, and also to change the sample frequency
8. Modify the widget used to request the measurement channel; replace it by a set of radio buttons, with labels “A1”, “A2”, “IN1”, “IN2”, “SEN”, etc.
9. Change the set of radio buttons to a set of checkboxes, so users can select up to four of them, and up to four channels can be sampled simultaneously
10. Provide a second plot widget, which will display the power spectrum of the signal (i.e. amplitude vs. frequency) after a Fast Fourier Transform

Propositions of solutions for this set of exercises are published in the “repository”.

5 Writing the user interface for a new experiment

5.1 Description of the new experiment

This experiment is about oscillations of a pendulum, and in most cases it can already be managed by an application distributed with expEYES Junior, known as “*Pendulum Waveform*”. You can launch the application already distributed within the package **expeyes**: while the **CRO+** application is running, click on the button “*EXPERIMENTS*”, then choose: *Pendulum Waveform*.

However, the program which implements currently “*Pendulum Waveform*” misses some features:

- it is built upon the graphic library *Tk*, which does not support easily right-to-left text writing in widgets, and has old-fashioned widgets
- it always plots the oscillations of the pendulum in non-real time, *i.e.* the voltages samples are taken when the computer asks for them. In most cases it is not an issue, but it may become one when the frequency of the oscillation is rising. At higher frequencies, the task scheduler of the computer’s operating system may prevent getting enough samples during one oscillation, which would lead to an odd-shaped curve.

5.1.1 Schematics

A massive object is hung under a spring. The top of the spring is tightly fixed, and a vertical insulated wire is hung just below the massive object. The end of this wire is unwrapped, so it acts as the cursor of a potentiometer. Eventually, some light sheet of plastic or of metal is tightened to the massive object in order to provide friction while it moves in the air. More friction can be added if a copper disk can be fixed along the vertical insulated wire underwater.

The potentiometer’s body is a measuring cylinder filled with copper sulfide (*concentration* = $0.01 \text{ mol} \cdot \text{L}^{-1} \simeq 1.6 \text{ g} \cdot \text{L}^{-1}$), which bears two circular electrodes, made with one turn of copper wire: one at the bottom of the cylinder, the other one near the top of the copper sulfide solution.

Both electrodes are connected through insulated wires to a 4.5 V battery. They can also be connected to two sockets of *expEYES Junior*: *GND* for 0 V

and *OD1* for $+5V$; in order to have the right voltage at *OD1*, the program must contain a line of code like `self.p.set_state(10,1)`, to enforce the logical *HIGH* state at *OD1*. Whenever one connects the battery, or sets power through the output *OD1*, she should consider which electrode needs to gain more copper, and connect the negative end of the battery to this electrode (copper is deposited on the cathode, while it is consumed at the anode).

When some current flows between both electrodes, it creates quite plane equipotential surfaces, regularly spaced vertically between the electrodes. So when the height of the vertical wire's tip swings up and down, its potential is a linear function of its height. Figure 27 summarizes the schematics.

5.1.2 Features wanted for the program

ExpEYES Junior comes with a program named “Data Logger” which can be used to monitor any signal captured by the measurement box. However, our experiment has a few unique features which call for a more appropriated program:

- both hands of the person who does the experiment may be busy just when the capture of data should start, *i.e.* when the pendulum begins its first oscillation. Generally, the person pulls slowly the elastic pendulum off its equilibrium point, takes care that the start position is well aligned with the axis of the main oscillation, and let it go suddenly. Then, the capture should begin instantly.
- depending on the experiment, one may be interested by more than 1800 measurements; 1800 measurements is the maximum number of voltage values which can be stored in the measurement box and sent as a response to a single query. From a didactic point of view, it is also interesting to plot the signal's curve immediately, without waiting the end of the measurement series.
- the program can provide additional facilities, like fitting the results with a predefined model, emphasizing some calculable results like a friction coefficient.

5.2 Implementing new features

Here is a reminder of wanted features:

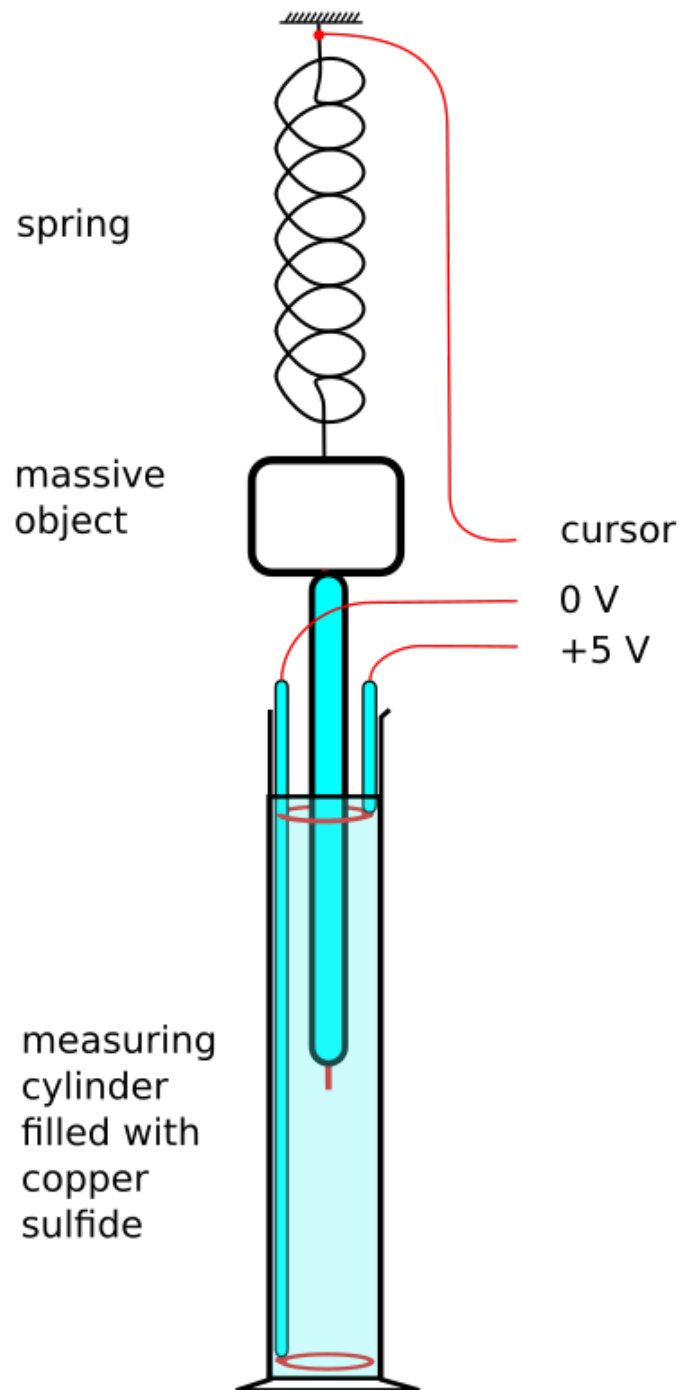


Figure 27: An oscillator made with a spring, and its liquid potentiometer

1. a button to wake up the program, so it will try to guess the start of the experiment
2. a button to stop recording the experiment
3. buttons to save sets of data, quit, etc.
4. when the program is in “*alert mode*”, it monitors the voltage input regularly, and begins to record data when the voltage changes quicker than a predefined threshold
5. when the program is no longer in “*alert mode*”, the user can launch fitting tools and compare the record to theoretical descriptions.
6. The curve of position vs. time is plotted inside a big frame, synchronously or asynchronously.

5.3 Coding the new program

5.3.1 Designing the user interface

With the program `designer`, let us create something like the structure displayed in figure 28.

The file is available from our repository, as `oscill.ui`. You can download it, and make a few modifications, like

- changing some part of the layout,
- changing the tooltips associated with some widgets,
- adding some new widgets (it is better to keep existing widgets).

You can notice that every widget which must be connected to the main program was named with an easy to remind identifier.

5.3.2 Exercise:

1. Modify the user interface in order to have all the control widgets stacked on its right. Save the new interface file with a new name. Figure 29 gives an outlook of a possible layout.
2. Modify the user interface to translate “manually” texts and tooltips to your language. Save it under a new name. *Please notice that this is not the recommended method to translate an application’s user interface. There are dedicated tools to perform this job more efficiently, and to have it done by other people.*

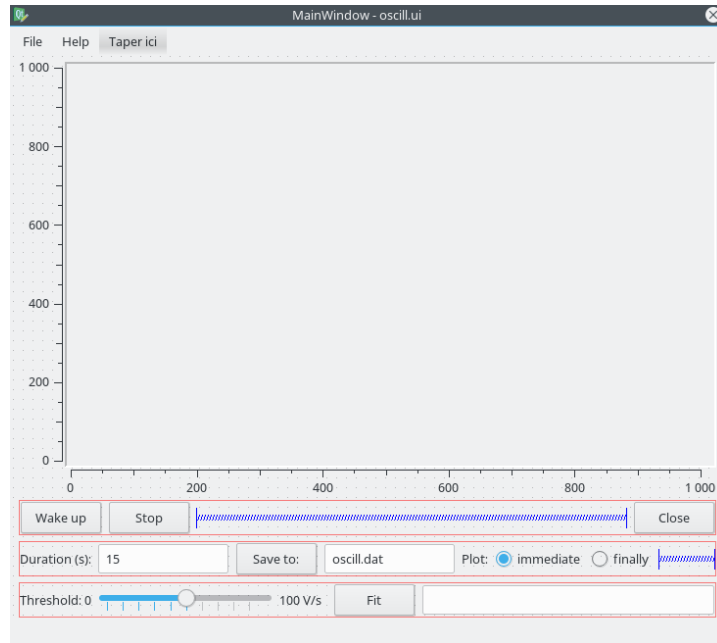


Figure 28: the user interface inside the application **designer**

5.3.3 Minimal structure of the main program

The main program must at least:

1. import necessary modules and the user interface module
2. define a custom class for the main window, which will be linked to the user interface
3. build the user interface upon initialization of the main window
4. connect signals managed by the widgets to methods of the main window
5. create the application object, instantiate the main window, and let it run.

This is done in the program `oscill11.py` which can be downloaded from the "repository"; figures 30 to 34 show how those five parts of the minimal program are implemented.

This program can be run, it raises no error. However, it makes nothing useful for the end-user so far. When you click buttons or trigger menu

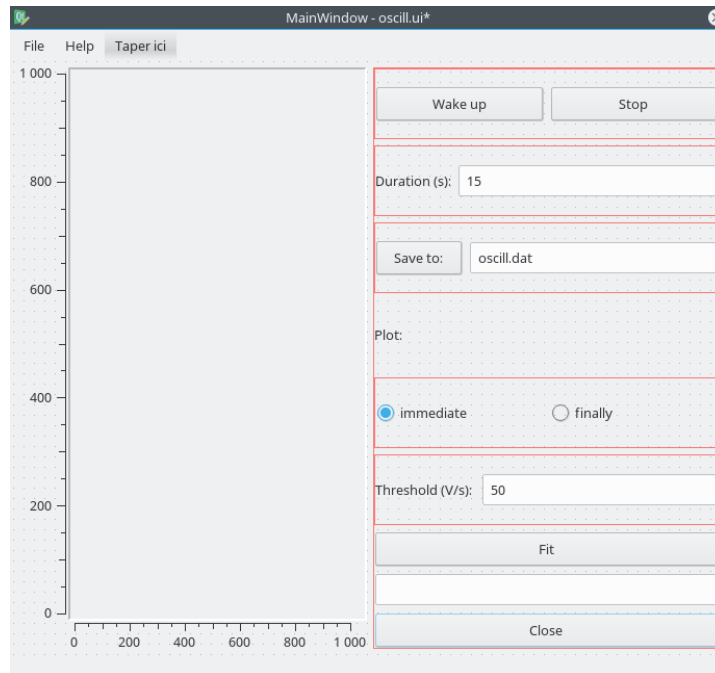


Figure 29: Control widgets of the application are stacked on the right

```

1  #!/usr/bin/python3
2
3  import sys
4  from PyQt4 import QtGui, QtCore
5  from UI_oscill import Ui_MainWindow
6  from qwt import QwtPlotCurve

```

Figure 30: importing modules and classes from modules

```

7
8  class MyWindow(QtGui.QMainWindow):
9      def __init__(self, parent=None):
10         QtGui.QMainWindow.__init__(self, parent)

```

Figure 31: the new class for the main window

```

11         self.ui=Ui_MainWindow()
12         self.ui.setupUi(self)

```

Figure 32: building the user interface

```

13         # connect methods to buttons' click signals
14         self.ui.wakeUpButton.clicked.connect(self.wakeUp)
15         self.ui.stopButton.clicked.connect(self.stop)
16         self.ui.closeButton.clicked.connect(self.close)
17         self.ui.saveButton.clicked.connect(self.save)
18         self.ui.immediateButton.clicked.connect(self.immediate)
19         self.ui.finalButton.clicked.connect(self.final)
20         self.ui.fitButton.clicked.connect(self.fit)
21         self.ui.action_Save_Ctrl_S.triggered.connect(self.save)
22         self.ui.action_Quit_Ctrl_Q.triggered.connect(self.close)
23         self.ui.actionManual.triggered.connect(self.manual)
24         self.ui.actionAbout.triggered.connect(self.about)
25         return
26
27     def notImplemented(self):
28         msg=QtGui.QMessageBox(QtGui.QMessageBox.Warning,"Sorry",
29                               "not yet implemented",)
30         msg.exec_()
31         return
32
33     wakeUp=stop=save=immediate=final=fit=manual=about=notImplemented

```

Figure 33: connecting signals to methods

```

35
36 if __name__=="__main__":
37     app=QtGui.QApplication(sys.argv)
38     window=MyWindow()
39     window.show()
40     sys.exit(app.exec_())
41

```

Figure 34: making it run

actions, a message pops up: “Sorry, not implemented”, except for the method `close` which is inherited from the ancestor `QMainWindow` class.

5.3.4 Adding methods to toggle the way curves will be plotted

There will be two methods to plot the curves:

1. immediately, while the pendulum is still oscillating
2. once, when the scheduled data have been recorded

To implement this, a new property, `self.isImmediate` is initialized to `True` in the main window’s constructor, and methods `immediate` and `final` are defined to change its value; see figure 35. Please notice that the identifiers `immediate` and `final` are no longer assigned to `notImplemented`.

5.3.5 Implementing naively the capture of data and the plotting routine

In this non-definitive version, the method `wakeUp` begins to capture data as soon as the button is clicked, and the flag `self.isImmediate` is disregarded: data are captured first, and the plot is refreshed later.

```

24         self.ui.actionAbout.triggered.connect(s
25         # custom properties
26         self.isImmediate=True
27         return
28
29     def immediate(self):
30         self.isImmediate=True
31         return
32
33     def final(self):
34         self.isImmediate=False
35         return

```

Figure 35: managing the plotting mode

Most code have been copied from the file `p3.py` studied previously. The implementation is done in file `oscill2.py`, which can be downloaded from the "repository". Figure 36 shows the code of the naive version of the method `wakeUp`.

```

52
53     def wakeUp(self):
54         # get the duration of the experiment in  $\mu$ s
55         duration = 1e6 * float(self.ui.durationEdit.text())
56         samples = 1800 # maximum sample number with 8 bit precision
57         # ensure that samples * delay will be slightly bigger than duration
58         delay=1+int(duration/1800)
59         t,v = self.p.capture(1,samples, delay)
60         self.curve.setData(t,v,len(t))
61         # display the result
62         self.ui.qwtPlot.replot()
63         return

```

Figure 36: implementation of the method `wakeUp`

When we try to launch the program `oscill2.py`, measurements are well done for durations of a few seconds: one, two or three seconds are well managed, but when the duration of measurements is four seconds or more, the box replies no longer and some error messages are emitted. One must unplug and plug again the USB cable to get a working system. The programmer's manual states that too short durations are not possible (there must be at least 4 μ s between successive samples), but issues with counter override are not documented.

So, we shall estimate that when the duration of the experiment must be longer than 3.5 s, the measurements must be done in "immediate" mode.

5.3.6 Less naive implementation for the plotting routine

This implementation can be found in the program `oscill3.py`, available from the "repository". The duration is tested: if it is longer than 3.5 s,

the “immediate” mode is enforced, and a timer is initialized to measure the signal regularly. As the computer does not work in real time, a timestamp must be made as soon as a new value is read. If the duration is in the range from 0.5 to 3.5 s, the mode is selected by taking in account radio buttons, and if the duration is shorter than 0.5 s, the “final” mode is enforced. The most relevant elements of code are shown in figure 37.

```

61 def wakeUp(self):
62     # get the duration of the experiment in s
63     duration = float(self.ui.durationEdit.text())
64     if duration < 0.5: # "final" mode is mandatory
65         self.ui.finalButton.setChecked(True)
66         self.isImmediate=False
67     elif duration > 3.5: # "immediate" mode is mandatory
68         self.ui.immediateButton.setChecked(True)
69         self.isImmediate=True
70     self.ui.qwtPlot.setAxisScale(QwtPlot.xBottom, 0, duration)
71     if self.isImmediate:
72         now=time.time()
73         self.t=[]
74         self.v=[]
75         self.curve.setData([],[],0)
76         self.startTime=now
77         self.stopTime=now+duration
78         # now the curve will grow until time.time >= self.stopTime
79         # thanks to self.timer's timeout events
80     else:
81         samples = 1800 # maximum sample number with 8 bit precision
82         # ensure that samples * delay will be slightly bigger than dur
83         delay=1+int(duration*1e6/1800)
84         t, self.v = self.p.capture(1,samples, delay)
85         self.t=[1e-3*date for date in t] # convert ms to s
86         self.curve.setData(self.t, self.v, len(self.t))
87     return

```

Figure 37: the new `wakeUp()` method

Figures 38 and 39 show how the “tick” method implements gathering voltage and time when the application gets a `timeout` signal, every 50 milliseconds, and how this timer was created during the construction of the main window.

```

88 def tick(self):
89     """ Callback for the timeout events """
90     t=time.time()
91     if t < self.stopTime:
92         v = self.p.get_voltage(1)
93         self.t.append(time.time()-self.startTime)
94         self.v.append(v)
95         self.curve.setData(self.t, self.v, len(self.t))
96     return
97
98

```

Figure 38: implementation of `tick`

5.3.7 Implementing the callback for the Stop button

When one clicks the *Stop* button, she wants to interrupt an acquisition of data. Interrupting this acquisition is not possible in “*final*” mode, but during

```

28         self.ui.actionABOUT.triggered.connect(self.about)
29         # create a timer
30         self.stopTime=time.time()
31         self.timer=QtCore.QTimer()
32         # connect the timer to the "tick" callback method
33         self.timer.timeout.connect(self.tick)
34         # 20 times per second
35         self.timer.start(50)
36         # ...

```

Figure 39: initializing the application’s timer

the measurement timespan in “*immediate*” mode, it can be allowed. Figure 40 shows the implementation of the `stop` callback method.

```

60         # ...
61         def stop(self):
62             # in "final" mode, this has no effect
63             # in "immediate" mode, it forces the plot to
64             # stop at the next tick call.
65             self.stopTime=time.time()
66             return
67

```

Figure 40: implementing the `stop` callback method

5.3.8 Implementing the callback for the Save button

This part is straightforward: relevant data are `self.t` and `self.v`, the callback method opens a text file and writes formatted data into it. See figure 41.

```

68         def save(self):
69             filename=self.ui.fileNameEdit.text()
70             with open(filename,"w") as outfile:
71                 for i in range(len(self.t)):
72                     outfile.write("{} {} \n".format(
73                         self.t[i], self.v[i]
74                     ))
75             self.ui.statusbar.showMessage(
76                 "Saved data to {}".format(filename), 3000 # 3 seconds
77             )
78             return
79

```

Figure 41: implementing the `save` callback method

5.3.9 Implementing the expectation of a rise in the slew rate

You remember that one of our concern was to make it easy for a user which has both hands in the experiment to launch the measurements. One way to achieve it is to begin measurements not when the “wake up” button is clicked, but later, when the voltage begins to swing. Therefore a control widget is designed to choose the desired slew rate threshold which will trigger

the measurement series. This widget is a slider, to select a threshold ranging from zero to some volts per second. The precise value of the higher selectable slew rate depends on the experiment's scheduled duration. The shorter it is, the higher the threshold slew rate should be.

Therefore, the “`textChanged`” signal of the duration edit line is connected to a callback function which changes the axis scale, but also changes the slider's displayed range.

```

60 def durationChanged(self, value, ampl=0):
61     """
62     Callback function for changed in ui.durationEdit
63     @param value the widget's value in case of an event
64     @param ampl an amplitudes (defaults to 0)
65     """
66     try:
67         duration=float(value)
68     except:
69         return
70     # set the axis scales for the plot widget
71     self.ui.qwtPlot.setAxisScale(QwtPlot.xBottom, 0, duration)
72     # draw the "zero" line
73     small=duration/1e6
74     self.curve0.setData([0, small, 2*small, 3*small, duration],
75                        [0, ampl, -ampl, 0, 0], 5)
76     # update the threshold rate
77     self.maxThreshold=150/duration
78     self.ui.thresholdLabel.setText("{} V/s".format(self.maxThreshold))
79     return
80

```

Figure 42: the `durationChanged` callback method

Then, some modifications are applied to the `wakeUp` method: a few lines are prepended just before the measurement routines, to call the `waitForThreshold` method.

```

138 # wait until the slew rate is fast enough
139 threshold = self.ui.thresholdSlider.value()*self.maxThreshold/100
140 self.waitForThreshold(threshold, duration, timeOut=5)
141 # start measuring
142 if self.isImmediate:
143     now=time.time()

```

Figure 43: a few lines prepended before the measurement calls

The `waitForThreshold` method has a parameter named `timeOut`, which allows one to specify the longest possible wait period.

5.3.10 The fit utility

Thanks to the library `expeyes.eyemath`, fitting a signal against a damped sinusoidal function is quite straightforward. The function `fit_dsine` does all the job, in a few steps: first, it guesses the pseudo-frequency of the signal thanks to a Fourier transform, then it calls a routine from the module `pylab.optimize` to minimize the differences between the signal and a parametric equation of a damped sinus. The parameters returned by the

```

107
108 def waitForThreshold(self, threshold, duration, timeOut=None):
109     """
110     wait for the input to change quickly enough
111     @param threshold a minimal voltage slew rate (V/s)
112     @param duration the duration of scheduled measurement series
113     @param timeOut the longest wait time (defaults to None)
114     """
115     start=time.time()
116     delay=int(duration/1000*1e6) # thousandth of duration, in µs
117     if delay < 4:
118         delay=4
119     t, v = self.p.capture(1, 2, delay)
120     slewRate=(v[1]-v[0])/(t[1]-t[0])*1000
121     while abs(slewRate)<threshold:
122         if timeOut != None and time.time()-start>timeOut:
123             return
124         t, v = self.p.capture(1, 2, delay)
125         slewRate=(v[1]-v[0])/(t[1]-t[0])*1000
126     return
127

```

Figure 44: the `waitForThreshold` method

modelling routine are fed back in a text field (the optimized model equation is displayed), and three additional curves are plotted in color: the model, and both positive and negative envelopes which encompass its values. Figures 45 and 46 show the program's source and a screenshot of the curves.

```

211
212 def fit(self):
213     """
214     Fitting data in self.t, self.v with a damped oscillation model
215     """
216     # fitting is performed by eyemath (aka em) thanks to
217     # scipy.optimize, and the error function defined by
218     # the module eyemath (line 92):
219     # p[0] * sin(2*pi*p[1]*x+p[2]) * exp(-p[4]*x) - p[3]
220     # so the vector of parameters is:
221     # amplitude, frequency, phase, DC average, damping factor.
222     yfit, plsq = em.fit_dsine(self.t, self.v, mode="Hz")
223     # display the fitting model
224     msg="{0:4.2f}*sin(2*pi*{1:4.2f}*t+{2:3.1f}))*exp(-{4:4.2f}*t)+{3
225         *plsq
226     )
227     self.ui.fitEdit.setText(msg)
228     # display three curves : model and model's envelopes
229     t=np.array(self.t)
230     f1=np.array(yfit)
231     f2=plsq[0]*np.exp(-plsq[4]*t)
232     f3=-1.0*f2
233     average=plsq[3]*np.ones(len(t))
234     red=QtGui.QColor("#ff0000")
235     self.fitCurve1.setPen(red)
236     self.fitCurve2.setPen(red)
237     self.fitCurve3.setPen(red)
238     self.fitCurve1.setData(t, f1, len(t))
239     self.fitCurve2.setData(t, f2+average, len(t))
240     self.fitCurve3.setData(t, f3+average, len(t))
241     return

```

Figure 45: source of the `fit` method

5.3.11 The two help methods

Most of the contextual help for the end user is defined during the design of the user interface. The program **designer** allows you to attach a help string to every widget, which will appear as tooltip popup when the mouse cursors

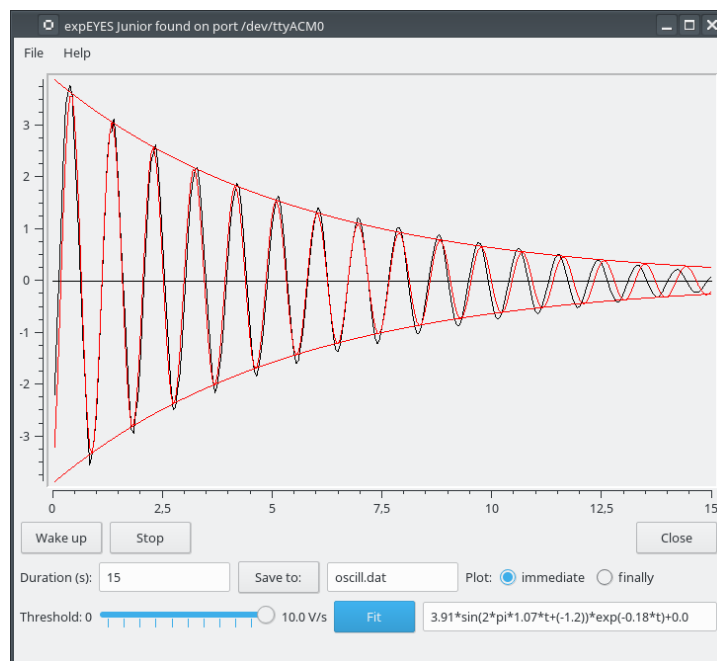


Figure 46: captured signal and its model

hovers it a few time. Other parts of the contextual help can be fed to the user via the status bar, and this can be written also while designing the user interface.

Two additional help methods are provided, respectively for menu items “Help → About” and “Help → Manual”. Both raise a browser window, which is fed a web page (respectively `license.html` and `oscillo4.html`). The first file is automatically made on the fly, based on the `license` variable defined in the program (at the begin of the code).

5.3.12 Access to the complete source code

Please visit the “repository”. The program is distributed under a GPL v3 license, hence you can copy it, modify and redistribute it, provided you comply with a few rules:

- redistributed versions must be ruled by the same license;
- authors must be quoted;
- if somebody received some *binary* version of the program, and asks you to provide her its source code, you must give access to it.

This source code can be easily modified, improved, extended, changed to manage other experiments, etc. For a beginner, three hundred lines of code are a big deal, but they are lighter than many other similar programs written with other languages and libraries. Copying and pasting parts of this program can speed up drastically the writing of other scientific programs.

For more experimented programmers, some parts of this code can be turned into prototype **classes**, which can be derived later into various usable end-user programs to control scientific experiments with expEYES. Exactly like the PyQt4 library allowed us to derive a special **class** `MyWindow` from the base **class** `QtGui.QMainWindow`, one can design a class featuring many widgets and methods reusable by various scientific programs, like the plot widget, the timer to periodically measure some signal, the detection of a slew rate as a synchronization tool, etc.

5.4 Exercises

Here a a few exercises which can be used to rework the program `oscill4.py`, the user interface `oscill4.ui`, the help file `oscill4.html`. To start any exercise, you can make a copy of all those files. Some other files like as `Makefile` and `oscill4.org` can also be of some help if you are using appropriate tools:

respectively the command `make` and an integrated development environment with `emacs`, `org-mode`, `TEXlive`.

1. Display a message in the status bar (`self.ui.statusBar`) just after one clicks on the *Wake up* button, and remove the message when the measurements begin. The message can be “waiting for a signal variation ...”
2. Make the previous message more interactive: format it with the template “{ } seconds left, waiting for a signal variation ...”; the time value at the begin of the message should be refreshed every second while the measurements have not yet begun. The initial time value is the `timeOut` which is a parameter of the `wakeForThreshold` method.
3. The upper limit of slew rate selectable with the threshold slider is a little too low for many applications, so the synchronization behavior can be too picky, and one cannot easily start the experiment. Modify the way this number is computed, to make the application more usable.
4. The menu provides a **Save** item, but no **Load File**. Implement a *Load File* submenu, and make something useful out of it.
5. Add a new submenu *Load recent file ...* and make something useful out of it. It should provide a keyboard shortcut, and a quick method to cycle between recently measured series.
6. Modify the `fit` method. The user should be given a choice, for example, between a pure sinus model (no friction), an exponentially damped sinus model (fluid friction), and why not, a linearly damped sinus model (solid friction).
7. Improve the user manual : for example, add some theoretical considerations about the damped sinus mode and fluid friction. You can adapt some contents from Wikipedia, do not forget to quote the original article and credit its authors if you do so.
8. Add some widget to comment the measured data, when they are well fitted with a damped sinus: say whether the oscillation is under-damped, critically damped, over-damped.
9. Modify the user interface, wisely add some funny colors.
10. Add a dialog to ask the user whether she wants to save data (only if necessary) when she closes the application.

6 Writing documentation for the end user

We shall not talk about writing techniques in this course: they have existed for a long time, and they do not depend on technical knowledge. Every author can launch a word processor and begin describing experiments. Whether the document will be good depends on literary skills which are not the topic of this course.

However we shall talk about methods which allow a community of developers and users to interact, to maintain good documentation. The model which all of us already know is Wikipedia. One can discuss about the methods used by mediawiki, the engine under Wikipedia (strong and weak points of the syntax used to write pages), but we must agree upon the success of Wikipedia: it is currently the biggest collaborative work ever published. Today, the page <https://en.wikipedia.org/wiki/Wikipedia> states that there are more than 40 million articles in more than 250 different languages, and as of February 2014, it had 18 billion page views and nearly 500 million unique visitors each month.

6.1 Key features of an effective collaboration engine

The history of Wikipedia demonstrated the advantage of widely open collaborative schemes over a hierarchically driven organization: when the project began, it was the sister project of Nupedia, which aimed to become an inline web encyclopedia, with articles written by experts, and approved by pair-review. On January 10, 2001, Sanger proposed on the Nupedia mailing list to create a wiki as a “feeder” project for Nupedia.

We know the result: Nupedia was superseded by Wikipedia, and when one wants to get information about Nupedia, she is redirected to an article of Wikipedia explaining why this project exists no longer.

6.1.1 Separated edition of neighboring sections

One problem of collaborative work is the risk of collisions when two contributors want to modify the same text simultaneously. This problem is partly worked around when a document is divided in many sections and subsections. The mediawiki engine allows two contributors to edit separate sections of the same level concurrently without colliding.

6.1.2 Template support

When more than one person is authoring a book, authors need a mechanism to create self-consistent contents. For a traditional book, this is done during some regularly scheduled meetings. For Wikipedia, asynchronous procedures are preferred. The solution is to define a template for data which need some consistency, and to report bugs when contributors forget to use the template.

A good example is the set of articles about chemical molecules: every molecule can be described by a set of physic properties, like temperature of melting, boiling point, molecular weight, etc.

Let us take a look at Wikipedia's article about Aspirin. Here is an excerpt of the source code in mediawiki syntax which gives the normalized description of the molecule:

```
{{Infobox drug
<!-- (many missing lines) -->
<!--Chemical data-->
| C=9 | H=8 | O=4
| molecular_weight = 180.157 g/mol
| smiles = O=C(Oc1ccccc1C(=O)O)C
| StdInChI_Ref = {{stdinchcite|correct|chemspider}}
| StdInChI = 1S/C9H8O4/c1-6(10)13-8-5-3-2-4-7(8)9(11)12/h2-5H,1H3,(H,11,12)
| StdInChIKey_Ref = {{stdinchcite|correct|chemspider}}
| StdInChIKey = BSYNRYMUTXBXSQ-UHFFFAOYSA-N
| synonyms = 2-acetoxybenzoic acid<br />acetylsalicylate
<br />acetylsalicylic acid<br />O-acetylsalicylic acid
| density = 1.40
| melting_point = 135
| boiling_point = 140
| boiling_notes = (decomposes)
| solubility = 3
}}
```

This code is rendered as in figure 47. You can find the source of the template used to deal with *Infobox_drug* in a special page of Wikipedia; this template is rather complicated, and results from the work of many contributors. However, templates can be defined with more simplicity, and they are still a very powerful tool to enforce consistency in a book written by multiple authors.

	PDB
	Chemical data
	Formula C ₉ H ₈ O ₄
	Molar mass 180.157 g/mol
	SMILES [show]
	InChI [show]
	Physical data
	Density 1.40 g/cm ³
	Melting point 135 °C (275 °F)
	Boiling point 140 °C (284 °F)
	(decomposes)
	Solubility in water 3 mg/mL (20 °C)
	(verify)

Figure 47: the info box for Aspirin

6.1.3 Translation support

One strong feature of mediawiki is the support of a very versatile engine to manage translations of a document.

A first document must be written in a pivot language, usually English (or bad English); then this document is marked automatically to allow other people to do translations.

Later, translators who log into the service can see the list of translatable pages, browse statistics about which of them are already translated, which of them deserve proofreading, which of them were modified and may be out of date regarding translation, etc.

At any moment, logged in translators can begin to translate a new page, which is done section by section.

6.1.4 Not WYSIWYG, rather WYSIWYM

When one wants to author a paper, the first idea is to use a word processor. Most of them are *WYSIWYG*, which means “What You See Is What You Get”: at any time, when one types her text, she can see on the screen quite

the result she will get upon printing it to paper.

Strangely enough, the wiki engine, which relies on a non-WYSIWYG input system, has taken precedence over papers created with word processors. Its success is due to another idea: “What You See Is What You Mean”: the syntax used to write an article denotes precisely the role of every part of a text. By default, the role is *paragraph*, and special markup is used to denote other roles like *section*, *subsection*, *subsubsection* etc., *table*, *table row*, *table cell*, *cross reference*, *hyperlink*, etc.

Every wiki engine has its own syntax to express those roles; such syntaxes are worth a training, since they enable you to collaborate with great and vivid communities of authors, to a great project. Mediawiki’s syntax is particularly worth your attention, because of the success of this project.

6.1.5 Free software, open standards

The syntax of Mediawiki text is an open standard, whose definition is published completely at www.mediawiki.org. . . BNF. The engine of Wikipedia is known as **mediawiki**, it is published as a free software, distributed under GPL V2+.

To create a working server featuring a wiki with this engine, here is the simplest recipe: hire a server with a correct bandwidth to Internet, install the stable Debian distribution, then add the package **mediawiki**; browse the newly created service, reply the questions which will make its particular configuration, *et voilà*.

As Mediawiki works with free software and open standards, it can be thought as some *eternal* software: if the organization which develops and maintains it fails to keep on doing it properly, anybody with suitable skills can step forward and take over the project. Another important consequence is that anybody can *rely* on the specification of data formats which come with mediawiki. So, one can author independently software which interacts with the mediawiki engine, and enrich its features. Many contributions were made by people independently from Wikimedia Foundation. Those contributions are sometimes integrated to new versions of the main engine, and sometimes not.

6.1.6 From a mediawiki to a book

WikiBooks is a website maintained by the Wikimedia Foundation, which allows anyone to publish a book based on work published in Wikipedia: you learn to link the wiki pages which you want to a book structure, and you

ask to create the book. A PDF document based on your choice is composed by the web server, and you can download it at will. Alternatively, you can place an order to get it printed, bound into a book with a suitable cover.

If you prefer, for some reason, to work independently from the Wikimedia Foundation, this is still possible! A criterion to say that a software is free is the so-called “Desert Island Test”:

A good **test case for** whether a license is free [...] is whether a disconnected group of people on a desert island could distribute the software among themselves. In the vim **case**, they cannot. (For example, **if** the vim maintainer flies over the island and drops down a message saying “you_must_hereby_send_me_your_changes”, how are the people down below to comply?) The fact that the vim maintainer can send the request does not say anything about whether the people receiving it could reply.

Here is the recipe to create a book from a wiki, in the Desert Island case: you hire a computer with a good bandwidth to the local network of the island, you install the last stable Debian distribution from a DVD you brought with you when you went to the Island, then you install and configure the package `mediawiki`, and finally you install (or you distribute to your peers) the package `mediawiki2latex` and its dependencies. Then you and every peer having installed the package `mediawiki2latex` are able to turn a page of the common wiki into a book.

6.2 Exercises: make a small book out of mediawiki pages

1. first variant: use the service provided at WikiBooks;
2. second variant: install the package `mediawiki2latex` and make the small book locally, without using WikiBooks’ service;
3. third variant: learn advanced features of `mediawiki2latex` and make a short book with particular styles applied in specific cases. For example, you define a template `Experiment` for your pages, which will be rendered as a header of level 3 inside the service powered by mediawiki, and rendered with special features in the PDF document, like a custom background color, custom fonts, etc.;
4. Chose some pages from Mediawiki which have currently no translation

in your language, and translate them; then create a small book out of them in your language.

7 ExpEYES as a common good

The expEYES box is Open Hardware, its design is published under the Open Hardware License. It implies that in foreseeable future, this measurement box will always exist. If current suppliers cease to sell it now, we can hire any electronics manufacturing company to make new boxes with the same features.

So, being part of the community of users of expEYES makes some sense. Currently, this community has few visible activities. Most people using expEYES are living in India, and the communication through Internet is in English language.

7.1 Mailing list

There is a mailing list owned by this community, named after the previous name of the project: PHOENIX-PROJECT where *PHOENIX* stands for Physics with Home-made Equipment and Innovative Experiments, a project initiated more than ten years ago by *Ajith Kumar* in *Inter University Accelerator Centre, New Delhi*. In this mailing list, training sessions are announced regularly. Some messages are invitations to propose a talk at free software meetings, and most of the traffic is about technical questions.

You can freely subscribe to this mailing list.

7.2 Software repositories

The latest versions of source code for expEYES are available at github.com, in the repositories named `expeyes-programs` and `expeyes-doc`, which contain respectively source code and schematics for the first one, *User Manual* and *Programmer Manual* for the second one. Currently, manuals and the message strings of the software are available in English and French language. If we want to extend the community of users, new localizations of these documents and messages can be done to comply with languages used in schools.

7.3 Contributing to the software

Previous parts of this course were about mastering Python and writing a minimal application for expEYES. So far, there have been few authors contributing to this software. However, as this is free software, you can step forward as soon as you want, any help is appreciated.

7.3.1 Reporting bugs and feature requests

Few people know that Free Software can progress significantly when end-users write bug reports and send wishes for new features. Of course, as long as software developers are not hired by some company to maintain the software, they decide finally what they will do. However, while this interaction seems to be very loose, it has proven to be faster and better than the support got by individual end-users when they try to ask for the help promised on the card box sold by some companies.


The software package for expEYES is primarily distributed as a Debian package, so the right way to report bugs and wishes is the bug reporting system of Debian.

If you are using **expeyes-junior**, you are inside a Debian GNU-Linux system (unless other packages have been created), for example during a session initiated by a Live USB stick.

7.3.2 reportbug utility, with an email-enabled system

If your Linux system is fully enabled to send e-mails, the best way for sending bugreports is the **reportbug** utility. To file a bug, launch the command **reportbug**, then go through the successive steps (figures 48 to 51).

1. giving the name of the package: **expeyes**
2. you can have a look at previous bug reports targetting this package; if your report is new, go to the next step, giving a short title to your report;
3. write the full bug report, with the help of the template which is given;
4. choose the severity of your report in the range Important → Wishlist;
5. send the bug report.




Reportbug

Please enter the name of the package in which you have found a problem, or type 'other' to report a more general problem. If you don't know what package the bug is in, please contact debian-user@lists.debian.org for assistance.

expeyes

Figure 48: reportbug – giving the package's name




Reportbug

Briefly describe the problem (max. 100 characters allowed). This will be the bug email subject, so keep the summary as concise as possible, for example: "fails to send email" or "does not start with -q option specified" .

here is the short description

Figure 49: reportbug – writing a very short title




Reportbug

How would you rate the severity of this problem or report?

Option	Description
important	a bug which has a major effect on the usability of a package, without rendering it completely unusable to everyone.
normal	a bug that does not undermine the usability of the whole package; for example, a problem with a particular option or menu item.
minor	things like spelling mistakes and other minor cosmetic errors that do not affect the core functionality of the package.
wishlist	suggestions and requests for new features.

Figure 50: reportbug – choosing the severity



Reportbug

Submit this report on expeyes

Submit the bug report via email.

Don't submit the bug report: instead, save it in a temporary file (exits rep

Figure 51: reportbug – final step

7.3.3 Reports when you are unsure that your system can send e-mails

In the last step previously described, just under the button *Submit the bug* ..., there is another button to save the bug report into a file. Choose this button, save the report in some text file. Then you can sign into your preferred webmail site, and paste relevant parts of the text file to send the bug report.

7.3.4 Translating software into your language

As **expeyes** is a software which follows good practices defined by GNU Gettext, you can translate the software without having programming skills. All you need is to know your language and the topics covered by **expeyes**, *i.e.* physic experiments. If you are reading this course in English, you already have the last prerequisite skill. Here is the complete method:

1. Download the file **messages.pot**, available [here](#).
2. Open the file in a text editor, like Notepad++ or Geany, and save it under a new name, which accounts for the new language. For example, to make a Spanish translation, the name should be **es_ES.po** or **es.po**; **es_ES** means Spanish as spoken in Spain, likewise **es_AR** would mean Spanish as spoken in Argentina. If you want to make a Turkish translation, the name **tr.po** should be correct. The suffix **.po** must be used for *PO* files, which are translation files in the most suitable format for GNU Gettext.
3. For every **msgid** instance, you must define the **msgstr** translation. See below for an example.
4. Send the new file to the maintainer, so he can rebuild the package with the new translation, finally get the new package when it is ready, install it.
5. Enjoy.

```
#: ../expeyes/eyesj.py:149 ../expeyes/eyes.py:119 ../expeyes/mca.py:40
msgid "Check the connections."
msgstr "Bağlantılarını kontrol edin."
```

Figure 52: Example translation en → tr.

7.3.5 Translating within a collaborative environment

The best collaborative environment for such a work is a so-called “Etherpad”. Here is a method to create an Etherpad inside the free web service provided by the Mozilla Foundation:

1. Open the URL <https://public.etherpad-mozilla.org>
2. Click on the *New Pad* button (figure 53)
3. We are provided a newly created Etherpad. We can paste the contents of the file `messages.pot` in its main text area (figure 54).
4. The URL of the working pad must be sent to teammates to allow them to access it (see figure 55)
5. As soon as teammates access this URL, their nicknames (everyone can customize its own) appear in the top right area, associated with a specific background color.
6. Everybody can begin working of the document which is shared inside the main area. A *chat* area is available in the bottom right corner of the window, and we can enjoy it to share messages within the team.

When the collaborative work is finished in the pad, one can copy it and paste it to another text editor, save it in a local file, etc.

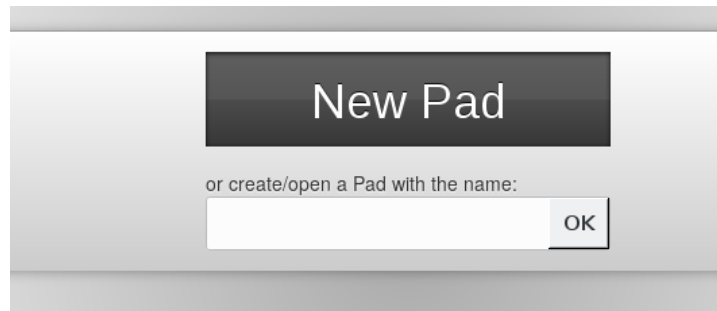


Figure 53: Creation of a new pad.

7.4 Exercises

1. Send a bugreport with severity *Wishlist* for the package `expeyes`. As the author of the course is also the maintainer of the package, you will

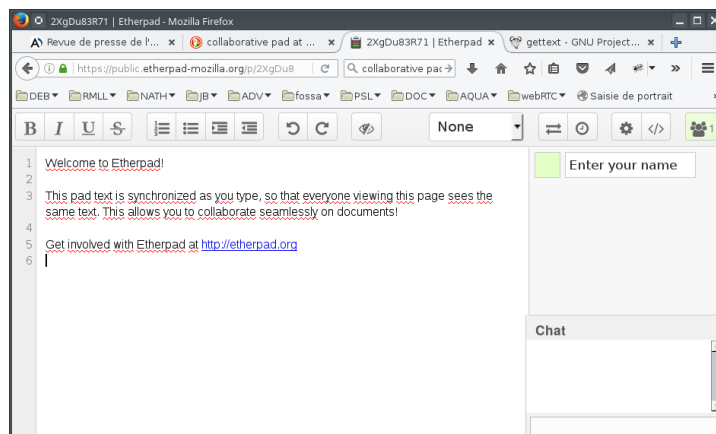


Figure 54: A newly created pad, with a welcome message (to be erased)

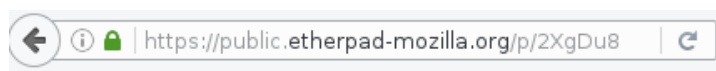


Figure 55: The pad's URL which must be shared

soon be able to watch how this bug report is managed. Do not hesitate to tell about feature requests.

2. Create a new Etherpad, for example in <https://public.etherpad-mozilla.org> and edit collaboratively a wishlist for possible continuations of this course.
3. Create a new etherpad and begin editing a *PO* file containing translations for your language.
4. Try to reopen a *PO* file in some specialized editor, like Kbabel, Linguist, PoEdit, or Emacs with the po-mode. Have a look at the additional features.

8 Conclusion, License

This very course is as free as are the programs and the hardware of expEYES. So you can distribute it, modify it (the source of the course is the file `course.org`, available in the github repository mentioned above, it is modifiable thanks to Emacs and org-mode.)

Do not hesitate to improve it, to translate it. Its license is Creative Commons-BY-SA version 3.0 (CC-BY-SA 3.0). So, here are your legal rights and obligations (summarized, see the full legal text when you want more details):

You are free to:

- **Share** — copy and redistribute the material in any medium or format
- **Adapt** — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

The author's e-mail address is: <georges.khaznadar@free.fr>