2

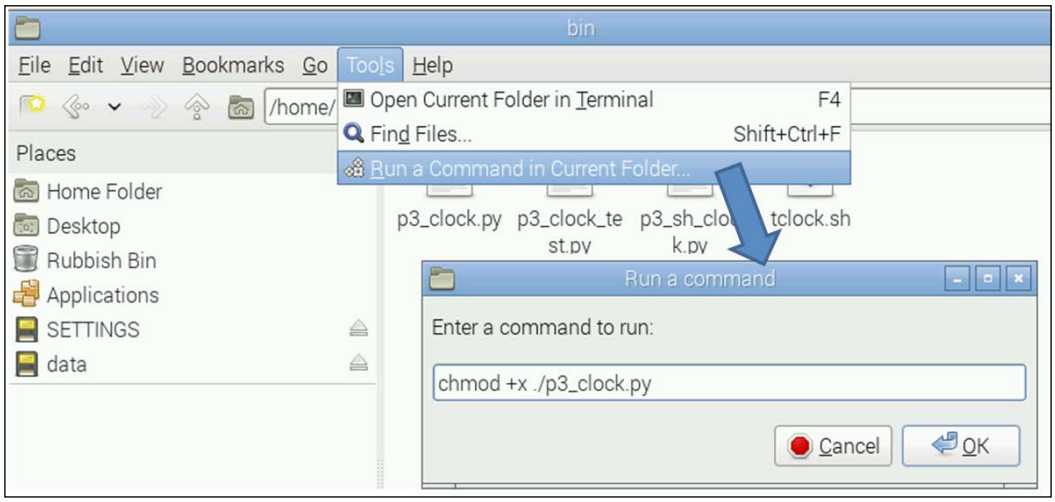
Project 1a – Developing the Talking Clock in Python 3

In Python Shell, select **File | New Window**; this will open a Python editor window. You can now type Python code into the editor window. The editor understands the rules for the code you type and will color code the text and manage indentation for you.

In the Python editor, perform the following steps:

1. Type `#!/usr/bin/python3`.
2. Select **File | Save As** and save the program as `p3_clock.py` to the directory `/home/pi/tclock/bin`**.**
3. Use **File Manager** to open the `/home/pi/tclock/bin` directory.
4. Click on **Tools | Run a Command in the Current Folder.**
5. In the **Run a Command...** form, enter `chmod +x ./p3_clock.py`, and then click on **OK** to set the file's executable property.

The following image captures the preceding steps:



The first line of text you just entered into the file is called a **shebang** string in Unix/Linux. Although it has nothing to do with Python programs, it is used by shell programs, such as Bash, to resolve the executable required for the program or script. When a file is executed and it starts with `#!`, the rest of the line is used to resolve the executable that handles the following text. The filename is appended as a command-line parameter. It is always a good policy to place the appropriate shebang in your Bash scripts and Python programs.

Now, we will implement our talking clock in Python code. With better programmatic control under Python, you won't need to use cron to start the program regularly and you can make the clock run more like a service.

Our talking clock had the following functionality as a Bash script implementation:

- Automatically start every minute (which was a cron function)
- Read the system time to get the local time on your Pi
- Produce a correctly formatted tts service time string
- Convert the time string to an audio stream
- Announce the time
- Exit (which simply waits for cron to start the script again)

We will respecify the functionality for a Python implementation:

- When started, run as a process with no user interface
- `*`-Get the local time
- Wait until seconds is `00`
- Produce a correctly formatted tts time string
- Call **mpg123** to get the time string returned as an MPEG audio stream announcement
- Sleep for 400 milliseconds
- Repeat from the step marked as `*`

In the Python editor window, type in the Python program code as shown here:

```
#!/usr/bin/python3

import os, time

#string variable for Google tts
gstring1="/usr/bin/mpg123 -q  'http://translate.google.com/translate_
tts?tl=en&q=The time is "
#variables
tsec1=""
time1=""
debug=False

#Code
while True:
    tsec1=time.strftime("%S")
    #if seconds roll over to 00 call audio
    if (tsec1 == "00"):
        time1 = time.strftime('%l:%M %p')
        gstring2=gstring1 + time1 +"'"
        if debug: print(gstring2)
        status=os.system( gstring2 )
    # loops every 400 milliseconds
    # to update the time
    time.sleep(0.4)
```

Starting from the top, here is a detailed explanation of the Python code:

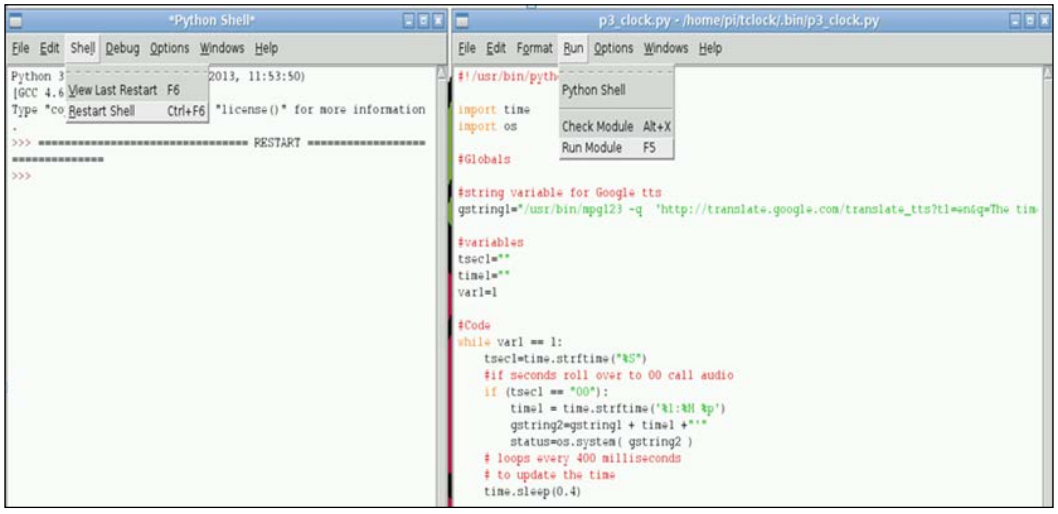
- The `import` statement allows the addition of libraries of additional functionality to be loaded by the interpreter. In this case, you loaded additional time and OS functions.
- Four variables are defined; three are string variables (quotes surround the strings) and one is a flag value (debug).
- The program really gets started at the `while` statement. This is a simple infinite loop since `True` is always `True`. Note that the `while` statement ends in a `":"` and all subsequent code is indented by four spaces; this is how Python shows loop and logic structures by indenting subsequent code.
- In the `while` loop, the variable `tsec1` is made equal to a string of the seconds value from the current time maintained by the OS.
- The `if` statement checks whether `tsec1` is equal to a string value of `00` and if it is, the code block indented under the `if` statement is executed. If it is not equal to `00`, then the block is skipped.
- If the string value tsec1 is `00`, then the four lines of code read the string values for the time and meridian, build the command string to get the tts audio stream, and call `os.system` to have this executed. Although the call returned a status value, we did not use it. There is also a line of code that prints out the `gstring2` value if `debug=True`.
- If the `tsec1` variable is not equal to `00` or if the inner block of code has finished execution, then the program calls `time.sleep` for `400` ms. This sleep period simply means that for most of each minute, the program will be asleep using very little CPU resource. However, it wakes to check the current value of the seconds count about twice per second, so it will never miss the rollover to `00`.

Running your Python clock in Python Shell

Now that you have entered the code into the Python editor window, you can start and run the application.

In the editor window, perform the following steps:

1. Click on **Run | Run module**, or simply select *F5* on the keyboard.
   This will bring the Python Shell window to the foreground, reset the shell, and then execute your program. The program will announce the time in less than a minute.
2. If you set **debug=True** in the program, then the string sent to the OS will be printed in the Python Shell window.
    - To stop the program in the Python Shell window, select Shell | Restart Shell or enter Ctrl + F6 on the keyboard. The screen image for these options is shown here:



Although you have completely rewritten the talking clock as a Python 3 application, you could have reused the Bash script you created to simplify the task. Look at the code for `p3_sh_clock.py` and see if you can work out what it is doing:

```
#! /usr/bin/python3

import time
import os

#variables
tsec1=""
```

```
#Code
while True :
    tsec1=time.strftime("%S")
    #if seconds roll over to 00 call audio
    if (tsec1 == "00"):
        status=os.system( "/home/pi/tclock/bin/tclock.sh" )
    # loops every 400 milliseconds
    # to update the time
    time.sleep(0.4)
```

This code is a very simple wrapper that does the timing and simply calls `tclock.sh` when it is time to announce the time.

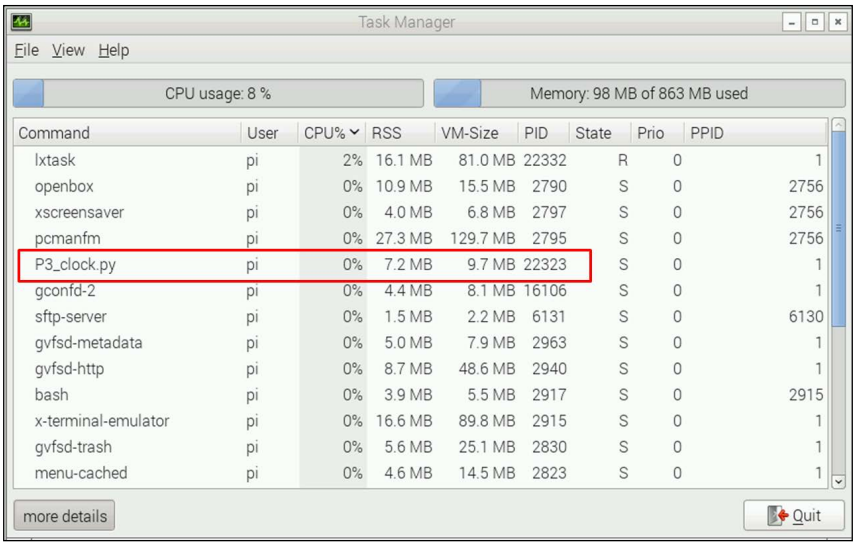
Reusing the Bash script implementation achieves the same goal, but the Python program replaces the cron job. This can be very useful if you have complex timing and schedules for a series of administrative or solution tasks, for example, perhaps the talking clock could have chimes for just the hourly and quarter-hourly messages or a more complex set of alarms. You are limited in what might be done when there is no user interface for the clock, but this could be added using one of several graphical interface packages that present a set of user controls.

Options for running your Python talking clock

Now that you have checked the functionality of the code for the talking clock, there are several options to start and run either of the programs.

You know how to start the clock from the Python editor using the *F5* function key and to reset using *Ctrl + F6* in the Python Shell window. Now, let's look at some other ways to start and stop the program:

1. Open **File Manager** and go to the `/home/pi/tclock/bin` directory and double-click on the file icon for `p3_clock.py`, which will raise a form providing an option to execute the file. This shows up because you set the executable flag for the file. Execute the file, and it will show up under **Task Manager**, as shown in the following screenshot:



2. In **Task Manager**, you can right-click and use **Kill** on the executable in the user process list to stop it. It will show up in the user process list because File Manager is running with user permissions.
3. Open a **Terminal** session in the `/home/pi/tclock/bin` directory and type `./p3_clock.py` to start it.
4. To stop the program, you can use *Ctrl* + *C* in the Terminal window. This method works because of the shebang at the beginning of the file, which tells the shell to use Python 3. In **Task Manager**, you can right-click and use **Kill** on the executable in the user process list to stop it. If you use `sudo ./p3_clock.py` to start it, then it will be in the root process list in Task Manager.

If you type `./p3_clock.py`, the session pauses until the program exits. You can exit from the Terminal session by selecting *Ctrl* + *C*, which also stops the program. You can also type `./p3_clock.py &`, and this causes the program to be started, but the session does not wait for it to exit. You get your command prompt back, but the program still *belongs* to your Terminal session, so if you close the window (terminate the session), then the clock program terminates too.

5. Open a Terminal window and type `Python3 /home/pi/tclock/bin/p3_clock.py`. To stop the program, use *Ctrl* + *C* in the Terminal session window.

The process is added to the **User** process list in **Task Manager**. This works because you provided two command-line parameters; the shell started Python 3 and was passed the file path to the Python program. When you look in **Task Manager**, you will see that it shows up as **python3** and not as `p3_clock.py`.

6. Open a Terminal window and type `sudo Python3 /home/pi/tclock/bin/p3_clock.py`. To stop the program, you can use *Ctrl* + *C* in the Terminal session window.

This method has a wrinkle to it; you won't see the process name in the user process list as it was started by the root (the superuser).

In Task Manager, you have to go to **View** | **Show Root tasks** to see it, and you cannot kill the program in the Task Manager window as you don't have root privileges. If you want to kill the process, look at the process ID number (**PID**), and, from a Terminal session command prompt, type `sudo kill xxxx`, where `xxxx` is the PID number.

Finally, if you want the program to start automatically without user interaction, you can use cron to start the program whenever the computer boots or reboots. Instead of using cron to specify a start time, or a regular start time, you can use a special flag to tell cron to start the program at boot time.

Unfortunately, the GUI you installed to make creating cron jobs easier does not understand this extension—so you have to use a special utility tool to add this information:

1. Open a Terminal services windows and type `sudo crontab -e` to open a **nano** editor window with the filename of the crontab in a `tmp` directory.
2. The window will show cron instructions as comments; scroll down to the first empty line after the last comment and type `@reboot python3 /home/pi/tclock/bin/p3_clock.py &` followed by the enter key.
3. Click *Ctrl* + *X* followed by `y` and *Enter* to save the file.

4. To see that you made no mistakes or to view the current crontab setting, you can type `sudo crontab -l`, which will print out the table entry.

At the command line, you typed `sudo crontab -e`, which edits the root user crontab; if you type `crontab -e`, you are editing the user crontab. Every user can have a separate crontab setting that makes it possible to load programs automatically for certain users as they log in or defines the crontab for root, which applies to all users.

5. You can now type `sudo reboot` in the terminal session or use the **Menu** | **Shutdown** option to reboot the system. When the system reboots, the talking clock should be started. Since you added it as the root user, it will show as **python3** in the root process list. If you edit the root crontab again, remove the `@reboot` command line, and reboot, otherwise the talking clock won't start.