```
**************************************************
  maXbox Starter 98
**************************************************
```

Solutions to run Python in a Box

```
------------------------------------
```

Max Kleiner

I rant, therefore I am.

When you run a Python script, the interpreter converts a Python program into something that that the computer can understand. Executing a Python program can be done in two ways: calling the Python interpreter with a shebang line, and using the interactive Python shell or to invoke the script file from a script shell editor like crontab or maXbox. I show 4 solutions.

Suppose you have a python script file httpheader1.py:
"G:\maXbox\works2022\maxbox4\examples\httpheader1.py"

```
               ____  ______.
  _____ _____  \   \/  /\_ |__   _______  ___
 /     \\__  \  \     /  | __ \ /  _ \  \/  /
|  Y Y  \/ __ \_/     \  | \_\ (  <_> >    <
|__|_|  (____  /___/\  \ |___  /\____/__/\_ \
      \/     \/      \_/     \/            \/
```

The best way to organize python-routines is to call it from a script. So you can log and control your scripts in time and structure. For example scheduling python scripts with crontab or maXbox as a script shell is fundamental when it comes to automating tasks using python. We will see how to schedule python scripts and pass the necessary parameters as well.
So in our httpheader1.py we just have to pass one argument, namely an URL to get some header information back:

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("-u", "--url", required=True,
            help="Full URL - http://www.freecybersecurity.org")
    args = vars(parser.parse_args())
    url = args["url"]
    print(funcHTTPHeader(url))
```

A first solution for returning the header is the function getDosOutput():

```
 println(getDosOutput('py examples\httpheader1.py --url
"https://www.ipso.ch/ibz"',exePath));
```

```
>>>
Cache-Control: max-age=3600, public
Content-Language: de
Content-Type: text/html; charset=UTF-8
Date: Thu, 11 Aug 2022 08:35:14 GMT
Etag: "1660206911"
Expires: Sun, 19 Nov 1978 05:00:00 GMT
Last-Modified: Thu, 11 Aug 2022 08:35:11 GMT
Link: <https://www.ipso.ch/>; rel="shortlink", <https://www.ipso.ch/ibz>;
rel="canonical", <https://www.ipso.ch/ibz/die-schweizer-schule-fuer-
technik-und-management>; rel="alternate"; hreflang="de",
<https://www.ipso.ch/ibz/die-schweizer-schule-fuer-technik-und-
management>; rel="revision"
```

```
Server: Apache/2.4.48 (Unix) OpenSSL/1.1.1k
Strict-Transport-Security: max-age=31536000; includeSubDomains
Vary: Cookie
X-Content-Type-Options: nosniff
X-Drupal-Cache: HIT
X-Drupal-Dynamic-Cache: UNCACHEABLE
X-Frame-Options: SAMEORIGIN
X-Generator: Drupal 8 (https://www.drupal.org)
X-Ua-Compatible: IE=edge
Connection: close
Transfer-Encoding: chunked
```

Another simpler one is the direct shebang line:

```
py examples\httpheader1.py --url "https://www.ipso.ch/ibz"
```

Or in Linux we have saved this script as httpheader1.py under our home directory, we can make it executable by entering the following command in a terminal:

```
$ sudo chmod +x httpheader1.py
```

Note: #!/usr/bin/python3 (specifying path of script interpreter at first line) is necessary if you wish to make the script executable.

```
#! /usr/bin/python3
```

To schedule this script to be executed, we need to enter a crontab scheduling expression into the crontab file. To do that, simply enter the following in a terminal:

```
  crontab -e
```

Then you might be prompted to select an editor, choose nano or maXbox and append the following line to the end of the opened crontab file:

```
*/2 * * * * /home/$(USER)/httpheader1.py --url"http:/.." >> /home/$(USER)/headeroutput.txt
```

A note to getDosOutput(): I have a commandline application coded in delphi that I need to call from a normal desktop application (also coded in delphi). In short, I want to call the commandline app and display the text it outputs "live" in a listbox or memo.
So theres a difference if it reads the output in one step or showing how the output can be read while the process is still running:

Second solution to call a Python script from the box is with the Python4Delphi engine:

```
   eg:= TPythonEngine.Create(Nil);
   eg.pythonhome:= PYHOME32;
   eg.opendll(PYDLL32)
   writeln(' ');
eg.execstr(filetostring('C:\maXbox\works2021\maxbox4\examples\httpheader1.py'));
   println(eg.evalstr('funcHTTPHeader("https://www.ipso.ch/ibz")'));
```

So with this function evalstr() takes an expression as an input and returns the result of the expression on evaluation in the box. The advantage in comparison with DosOutput is the access to the variables and members of the script in our script. The access can be a global or local scope in the context of the shell script.

The arguments are a string and optional globals and locals. If provided, globals must be a dictionary. If provided, locals can be any mapping object. That means we dont need a main function or a main part in the httpheader1.py we can call (after execute the script with eg.execstr) the function funcHTTPHeader direct.

```
eg.execstr(filetostring('C:\maXbox\works2021\maxbox4\examples\httpheader1.py'));
   println(eg.evalstr('funcHTTPHeader("https://www.ipso.ch/ibz")'));

>>> Cache-Control: max-age=3600, public
Content-Language: de
Content-Type: text/html; charset=UTF-8
Date: Thu, 11 Aug 2022 12:22:55 GMT
Etag: "1660220574"
Expires: Sun, 19 Nov 1978 05:00:00 GMT
Last-Modified: Thu, 11 Aug 2022 12:22:54 GMT
▯▯▯ mX4 executed: 11/08/2022 14:22:55  Runtime: 0:0:5.620  Memload: 47% use
```

Many programming languages have a special function that is automatically executed when an operating system starts to run a program, usually called main() and must have a specific return type and arguments according to the language standard. On the other hand, the interpreter executes scripts starting at top of the file, and there is no specific function that it automatically executes.

```
    args = vars(parser.parse_args())
    url = args["url"]
    print(funcHTTPHeader(url))    #return in main()
```

In this code, there is a function called main() that prints the return of the function funcHTTPHeader(url) when the Python interpreter executes it. In the sense of a shell script we import the code as a module. But whats the difference between script and module:

A script is a file that you intend to execute from the command line to accomplish a task.

A Python module is a file that you intend to import from within another module or a script, or from the interactive interpreter. In our case maXbox is the within another script. You can read more about modules in Python Modules and Packages – An Introduction.

https://realpython.com/python-main-function/

The last and 4. solution is the complete integration of a script within a script, sounds complicated but is more work than brain.

We define the whole script from the file as a integrated const in our box:

```
Const HTTP_HEADER_DEF =
'import urllib.request                              '+LF+
'import argparse                                    '+LF+
'############# HTTP Header ##############           '+LF+
'def funcHTTPHeader(url):                           '+LF+
'  agent= {"User-Agent":"Mozilla/5.0 (OS X 10.13) Gecko/201
Firefox/62.0"}'+LF+
'  req = urllib.request.Request(                    '+LF+
'          url,                                     '+LF+
'          data=None,                               '+LF+
'          headers=agent                            '+LF+
'          )                                        '+LF+
'  try:http = urllib.request.urlopen(req)                       '+LF+
'  except urllib.error.URLError as err:                         '+LF+
'      print("ERROR: {} {}".format(err.code,err.reason))     '+LF+
'  return http.headers                                       ';

   eg.Execstring(HTTP_HEADER_DEF);
   println(eg.evalstr('funcHTTPHeader("https://www.ipso.ch/ibz")'));
```

When the integrated interpreter python engine <eg.Execstring> reads a Python file or the const in our case, it first sets a few special variables of the scope. Then it executes the code from the file or the const in our script. Its almost the same as solution 3 with the difference of the script integration, solution 3 can als be a net share or remote file, solution 4 is integrated as const and can be edited direct in the box.

```
eg.execstr(filetostring('C:\maXbox\works2021\maxbox4\examples\httpheader1.py'));
   println(eg.evalstr('funcHTTPHeader("https://www.ipso.ch/ibz")'));
```

Python files are called modules and they are identified by the .py file extension. A module can define functions, classes, and variables. Again, when running file_one you will see that the program recognized which of these two modules is __main__ and executed the code according to our first if else statements:

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
```

When modules like this httpheader1.py with just one single function funcHTTPHeader are being imported and run, their functions will be imported, and top level code executed

Sort of a microservice you find at:
https://www.academia.edu/31112544/Work_with_microservice_maXbox_starter48.pdf

```
                       .,,uod8B8bou,,.
              ..,uod8BBBBBBBBBBBBBBBBRPFT?l!i:.
         ,=m8BBBBBBBBBBBBBBBRPFT?!||||||||||||||
         !...:!TVBBBRPFT||||||||||!!^^""'   ||||
         !.......:!?|||||||!!^^""'          ||||
         !.........||||                     ||||
         !.........||||   ##                ||||
         !.........||||                     ||||
         !.........||||                     ||||
         !.........||||                     ||||
         !.........||||                     ||||
         `.........||||                    ,||||
          .;.......||||               _.-!!|||||
   .,uodWBBBBb.....||||       _.-!!||||||||||!:'
!YBBBBBBBBBBBBBBb..!|||:..-!!|||||||||!iof68BBBBBb....
!..YBBBBBBBBBBBBBBb!!|||||||||!iof68BBBBBBRPFT?!::   `.
!....YBBBBBBBBBBBBBBbaaitf68BBBBBBRPFT?!:::::::::     `.
!......YBBBBBBBBBBBBBBBBBBBBRPFT?!::::::;:!^"`;:::       `.
!........YBBBBBBBBBBRPFT?!::::::::::^''...::::::;         iBBbo.
`..........YBRPFT?!::::::::::::::::::::::::;iof68bo.      WBBBBbo.
  `..........:::::::::::::::::::::::;iof688888888888b.     `YBBBP^'
    `........::::::::::::::::::;iof688888888888888888888b.    `
      `......:::::::::;iof688888888888888888888888888888b.
        `....:::;iof688888888888888888888888888888888899fT!
          `..::!8888888888888888888888888888888899fT|!^"'
            `' !!988888888888888888888888888899fT|!^"'
```

```
                `!!8888888888888888899fT|!^"'
                  `!988888888899fT|!^"'
                    `!9899fT|!^"'
                      `!^"'
```

Conclusion:

The idea of separating a script into another script is nothing new; there are other programming paradigms which address this same concept, such as Service Oriented Architecture (SOA) or POST-Services. Translation is the written transfer of a message from one language to another. Interpretation refers to oral translation, but can also mean interpreting the meaning and intention of the message. Translation is made from the source language to the target language.

You may also see Python scripts executed from within packages by adding the -m argument to the command. Most often, you will see this recommended when you're using

```
      pip: python3 -m pip install 'package_name'.
```

Adding the -m argument runs the code in the __main__.py module of a package.

We have a 4 solution step focus in this tutorial:
----

1. Call the script from terminal or shell:

```
    > py examples\httpheader1.py --url "https://www.ipso.ch/ibz"
```

2. Reroute the script to another output:

```
    > println(getDosOutput('py examples\httpheader1.py --url
"https://www.ibz.ch"',exePath));
```

3. Integrate the interpreter:

```
eg.execstr(filetostring('C:\maXbox\works2021\maxbox4\examples\httpheader1.py'));
    > println(eg.evalstr('funcHTTPHeader("https://www.ipso.ch/ibz")'));
```

4. Integrate the script and the interpreter:

```
    eg.Execstring(HTTP_HEADER_DEF);
    > println(eg.evalstr('funcHTTPHeader("https://www.ipso.ch/ibz")'));
```

Simplify API development for users, teams, and enterprises with the Swagger open source and professional toolset. Find out how Swagger can help you design, document your APIs at scale.

```
 >>>

               _od#HMM6&*MMMH::-_
          _dHMMMR??MMM? ""|  `"'-?Hb_
       .~HMMMMMMMMHMMM#M?        `*HMb.
     ./?HMMMMMMMMMMM"*""""           &MHb.
    /'|MMMMMMMMMMM'             -    `*MHM\
   /  |MMM'MMHHM''                    .MMMHb
```

```
  |   9HMMP    .Hq,                   TMM'MMH
 /     |MM\,H-""&&6\__                `MMMMMMb
|       `""HH#,       \              - MMMMMMM|
|           `HoodHMM###.              `9MMMMMH
|              .MMMMMMMM##\             `*"?HM
|         ..  ,HMMMMMMMMMMMo\.              |M
|             |MMMMM'MMMMMMM'MNHo           |M
|              ?MMMMMMM'MMMMMMMM*           |H
|.              `#MMMMMMMM'MMMM'           .M|
 \                `MMMMMMMMMMMM*           |P
 `\                MMMMMMMMT"'            ,H
  `\              `MMM'MMH?              ./
    \.            |MMMH#"               ,/
     `\.          |MMP'               ./'
       `~\        `HM:.-    .       ,/'
          "-\_       '_\ .      _.-"
              "-\-#odMM\_,oo==-"
```

```
 This is what we want to use in maXbox:

 const res = await fetch("https://libretranslate.com/translate", {
 method: "POST",
 body: JSON.stringify({
  q: "Hello!",
  source: "en",
  target: "es"
 }),
 headers: { "Content-Type": "application/json" }
 });
 console.log(await res.json());


Source of the integrate script at: 1150_pydemo42httpheader.txt

http://www.softwareschule.ch/examples/1150_pydemo42httpheader.txt
```

Each service should be independently developed and deployed. No coordination should be needed with other service teams if no breaking API changes have been made. Each service is effectively it
it's own codebase and lifecycle.

```
                    ----
                   /~@@~\,
 _______ . _\_\___/\ __ /\___|_|_ . _______
/ ____  |=|     \  <_+>  /      |=| ____ \
~|    |\|=|======\\_______//======|=|/|    |~
 |_   |    \      |      |      /    |    |
  \==-|     \     |  mX4 |     /     |----|~~)
  |   |      |    |      |    |      |_____/~/
  |   |       \_____\_____/_____/    /    / /
  |   |         {----------}       /_____/ /
  |____|       /~~~~~~~~~~~~\     |_/~|_|/
   \_/        [/~~~~~||~~~~~\]     /__|\
   | |         |    ||||    |     (/|[[\)
   [_]        |     |  |     |
              |_____|  |_____|
              (_____)  (_____)
              |     |  |     |
              |     |  |     |
              |/~~~\|  |/~~~\|
              /|___|\  /|___|\
```

```
             <_______><_______>

 A microservice architecture shifts around complexity. Instead of a single
complex system, you have a bunch of simple services with complex
interactions.

  Doc:
  https://www.freecodecamp.org/news/if-name-main-python-example/
  https://www.geeksforgeeks.org/crontab-running-a-python-script-with-
parameters/
  https://www.onworks.net/software/windows/app-maxbox
  http://www.softwareschule.ch/examples/1142_list_collections_pydemo42.txt

http://www.softwareschule.ch/examples/1145_271_closures_study_op_sys_tutor97.txt

  -----_____
                   _____------           __      ----_
            ___----                ___------             \
               ----________            ----               \
                           -----__    |              _____)
                                __-                 /     \
                    _______-----    ___--           \    /)\
              ------_______      ---____             \__/  /
                           -----__    \ --     _         /\
                                  --__--__      \_____/   \_/\
                                          ----|   /            |
                                              |  |____________|
                                              |  |  ((_(_)| )_)
                                              |  \_((_(_)|/(_)
                                              \              (
                                               \_____________)


namespace RegExApplication
    {
        class Program
        {
            private static void showMatch(string text, string expr)
            {
                int arex = 46;
                Console.WriteLine("The Expression: " + expr + ' '+arex);
                MatchCollection mc = Regex.Matches(text, expr);
                foreach (Match m in mc)
                {
                    Console.WriteLine(m);
                }
            }
            static void Main(string[] args)
            {
                string str = "A Thousand Splendid Suns";
                Console.WriteLine("Matching words that start with 'S': ");
                showMatch(str, @"\bS\S*");
                FileIOApplication.FileProgram.Mainfiler();
                BinaryFileApplication.Program.Mainbinary();
                WindowsFileApplication.Program.Mainwin();
               //
DelegateAppl.BoilerEventAppl.RecordBoilerInfo.Mainboiler();
                //DelegateAppl.TestDelegate.MainDelegate(["Arg 1","Arg 2",
"Arg 3"]);
                //  DelegateAppl.TestDelegate.MainDelegate("Arg 3");

                //System.Windows.Input.ICommand.Equals.
```

```
                    Console.ReadKey();
                    foreach (var arg in args)
                    {
                        Console.WriteLine(arg);
                    }
                    /* for(int i = 0; i < args.length; i++) {
                        Console.WriteLine(args[i]);
                        }
                    Console.ReadKey();*/
                    //205
                }
            }
        }

def generate_power_func(n):
    print "id(n): %X" % id(n)
    def nth_power(x):
        return x**n
    print "id(nth_power): %X" % id(nth_power)
    return nth_power

>>> raised_to_4 = generate_power_func(4)
id(n): CCF7DC
id(nth_power): C46630
>>> repr(raised_to_4)
'<function nth_power at 0x00C46630>'

def generate_power_func(n):
    print "id(n): %X" % id(n)
    def nth_power(x):
        return x**n
    print "id(nth_power): %X" % id(nth_power)
    return nth_power

>>> raised_to_4 = generate_power_func(4)
id(n): CCF7DC
id(nth_power): C46630
>>> repr(raised_to_4)
'<function nth_power at 0x00C46630>'

In maXbox4 a reroute with itself is also possible with -c:

  CaptureConsoleOutput('maXbox4 -c firstdemo.txt', memo2);

memo2 is the console (output) window in maXbox editor!

maXbox4 actual Version
Total of Function Calls: 35771
SHA1: 4.7.6.10 30bace36b037686509bbbee256e22daa52b3df70
CRC32: 500F69DD: 31.8 MB (33,400,088 bytes)
ZIP maxbox4.zip SHA1: 9DA15BFD72471108FCF746669C6AFD96E9A0E54C
```