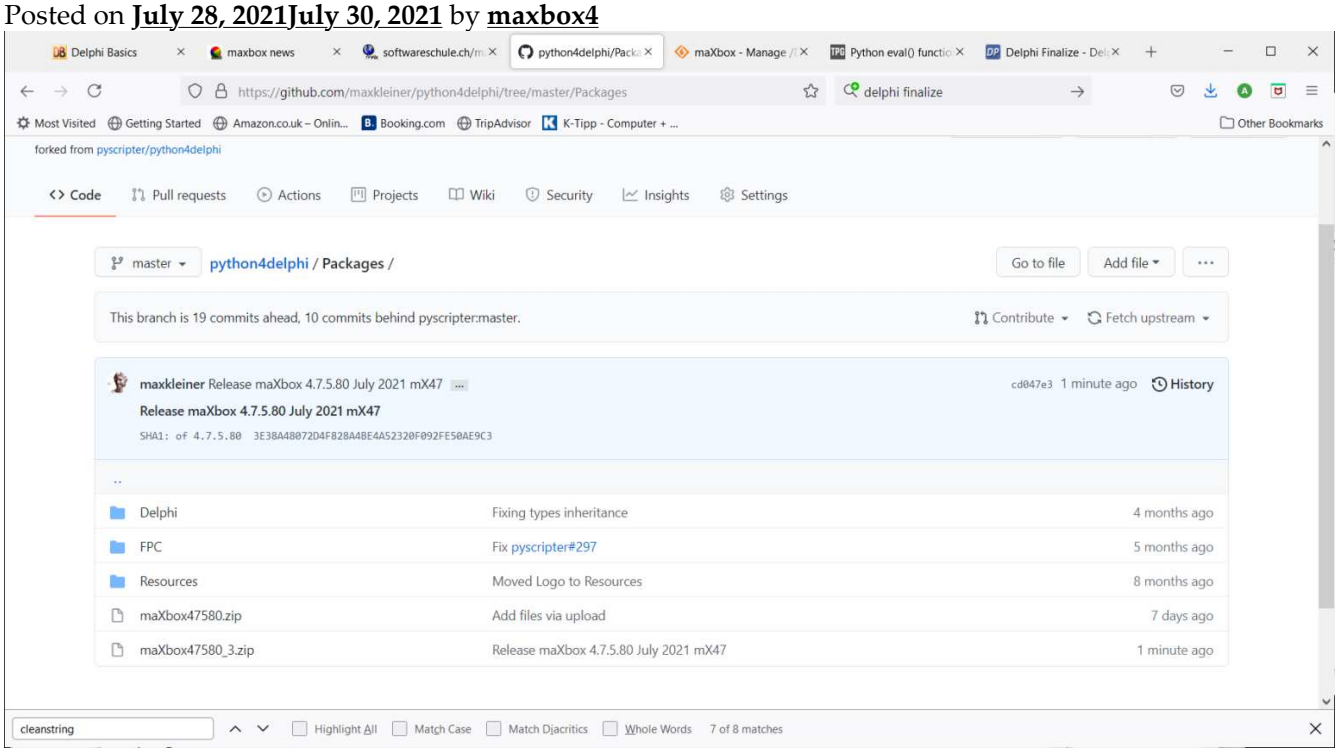
Welcome to maXbox Code (Blog)

maXbox

Python4maXbox Code

Posted on **July 28, 2021July 30, 2021** by **maxbox4**



Thanks to Python4Delphi we now can evaluate (for expressions) or exec (for statements) some Python code in our scripts. This version 4.7.5.80 July 2021 allows us with the help of a Python Dll and an environment with modules in site-packages execute Py-functions. But the most is only available in a 32-bit space, possible also with 64-bit Python means the call of the external shell (ExecuteShell) with installed Python versions to choose from. By the way also a **Python4Lazarus** is available.

1413 unit uPSI_PythonEngine.pas _P4D_Beta
1414 unit uPSI_VclPythonGUIInputOutput;
1415 unit uPSI_VarPyth;
1416 unit JclUsesUtils;
1417 unit uPSI_cParameters;
1418 unit uPSI_WDCCMisc; (uPSI_cFileTemplates);
1419 uPSI_WDCCOleVariantEnum.pas
1420 unit uPSI_WDCCWinInet.pas _WDCC
1421 uPSI_PythonVersions.pas _P4D_
1422 unit uPSI_PythonAction.pas _P4D_

Imagine you need a 512bit hash and you don't have the available function. SHA256 or SHA512 is a secure hash algorithm which creates a fixed length one way string from any input data. OK you start the Python-engine in your maXbox script and load the DLL. Most of the time you don't need to install Python cause you find a DLL for example in the Wow64 subsystem and load it. **WoW64** (**W**indows 32-bit **o**n **W**indows **64**-bit) is a subsystem of the **Windows (https://en.wikipedia.org/wiki/Microsoft_Windows) operating system (https://en.wikipedia.org/wiki/Operating_system)** capable of running **32-bit (https://en.wikipedia.org/wiki/32-bit)** applications on 64-bit Windows.

To get a Dll that fits your size you can check with

```
writeln('is x64 '+botostr(Isx64('C:\maXbox\EKON24\python37.dll')));
```

I provide one which we use most at:

https://sourceforge.net/projects/maxbox/files/Examples/EKON/P4D/python37.dll/download (https://sourceforge.net/projects/maxbox/files/Examples/EKON/P4D/python37.dll/download)

Search for registered versions is possible with the function *GetRegisteredPythonVersions : TPythonVersions');* On 64-bit Windows the 32-bit python27.dll is really in C:\Windows\sysWOW64. But if you try opening the C:\Windows\system32\python27.dll in a 32-bit process, it'll open just fine. If I'm not mistaken, WOW stands for Woodoo Of Windows. 😉

To make sure your path is the right test it with *OpenDll()*:

```
procedure TDynamicDll.LoadDll;
begin
  OpenDll( DllName );
end;

  eng.dllpath:= 'C:\maXbox\EKON25'
  eng.dllname:= 'python37.dll';
  eng.AutoLoad:= false;
  try
    eng.OpenDll('C:\maXbox\EKON25\python37.dll');
```

Or then you type the path and name of the Dll:

```
with TPythonEngine.create(self) do begin
 //Config Dll or Autoload
   Dllpath:= 'C:\Users\max\AppData\Local\Programs\Python\Python36-32\'
   Dllname:= 'python37_32.dll';
   LoadDll;
   writeln(pythonhome)
   writeln(ExecModule)
   pypara:= 'https://en.wikipedia.org/wiki/WoW64 (https://en.wikipedia.org/wiki/WoW64)';
   //pypara:= filetostring(exepath+'maXbox4.exe')
   try
     writeln(evalstr('__import__("math").sqrt(45)'));
     writeln(evalstr('__import__("hashlib").sha1(b"https://en.wikipedia.org/wiki/WoW64 (
     writeln(evalstr('__import__("hashlib").sha1(b"'+pypara+'").hexdigest().upper()'));
     writeln(evalstr('__import__("hashlib").sha256(b"'+pypara+'").hexdigest().upper()'))
     writeln(evalstr('__import__("hashlib").sha512(b"'+pypara+'").hexdigest().upper()'))
   finally
     free
   end;
 end;
```

So the last line is the sha512 and the result is: 8B94C64213CAD……and so on. The important thing is the evalstr() function. The `eval()` allows us to execute arbitrary strings as Python code. It accepts a source string and returns an object. But we can also import modules with the inbuilt syntax '**import**("hashlib").

The `eval()` is not just limited to simple expression. We can execute functions, call methods, reference variables and so on. So we use this by using the `__import__()` built-in function. Note also that the computed hash is converted to a readable hexadecimal string by hexdigest().upper()' and uppercase the hex-values in one line, amazing isn't it.

We step a bit further to exec a script in a script! If we call a file or an **const** Python command then we use ExecString(PYCMD); The script you can find at: **http://www.softwareschule.ch/examples/pydemo.txt (http://www.softwareschule.ch/examples/pydemo.txt)**

The essence is a bit of script:

```
const PYCMD = 'print("this is box")'+LB+
              'import sys'+LB+
              'f=open(r"1050pytest21.txt","w")'+LB+
              'f.write("Hello PyWorld_mX4, \n")'+LB+
              'f.write("This data will be written on the file.")'+LB+
              'f.close()';
```

The LB = CR+LF; is important cause we call it like a file or stream and exec() is cleaning (delete CR) and encoding the passing script afterwards:

```
writeln('ExecSynCheck1 '+botostr(eng.CheckExecSyntax(PYCMD)));
eng.ExecString(PYCMD);
```

We also check the syntax before to prevent an exception: Exception: Access violation at address 6BA3BA66 in module 'python36.dll'. or 'python37_32.dll' Read of address 000000AD. Free the engine means destroying it calls *Py_Finalize*, which frees all memory allocated by the Python Dll.

Or, if you're just using the Python API without the VCL wrappers like we do, you can probably just call *Py_NewInterpreter* on your *TPythonInterface* object to get a fresh execution environment without necessarily discarding everything done before!

By success of execute PYCMD a file (1050pytest21.txt) is written with some text so we executed line by line the **PYCMD**.

This is the whole tester Procedure PYLaz_P4D_Demo2; but key takeaway is that only use `eval()` with a trusted source.

https://thepythonguru.com/python-builtin-functions/eval/ (https://thepythonguru.com/python-builtin-functions/eval/)

https://github.com/maxkleiner/python4delphi/tree/master/Tutorials (https://github.com/maxkleiner/python4delphi/tree/master/Tutorials)

Eval can be evil if untrusted

```
Procedure PYLaz_P4D_Demo2;
//https://wiki.freepascal.org/Python4Delphi (https://wiki.freepascal.org/Python4Delphi)
var eng : TPythonEngine;
    out1: TPythonGUIInputOutput;
begin
  eng:= TPythonEngine.Create(Nil);
  out1:= TPythonGUIInputOutput.create(nil)
  out1.output:= pyMemo; //debugout.output;  //memo2;
  out1.RawOutput:= False;
  out1.UnicodeIO:= False;
  out1.maxlines:= 20;
  out1.displaystring('this string thing')
  //eng.IO:= Out1;
  Out1.writeline('draw the line');
  try
    eng.LoadDll;
    eng.IO:= Out1;
    if eng.IsHandleValid then begin
      writeln('DLLhandle: '+botostr(eng.IsHandleValid))
      WriteLn('evens: '+ eng.EvalStringAsStr('[x**2 for x in range(15)]'));
      WriteLn('gauss: '+ eng.EvalStringAsStr('sum([x for x in range(101)])'));
      WriteLn('gauss2: '+ eng.EvalStr('sum([x % 2 for x in range(10100)])'));
      WriteLn('mathstr: '+ eng.EvalStr('"py " * 7'));
      WriteLn('builtins: '+ eng.EvalStr('dir(__builtins__)'));
      WriteLn('upperstr: '+ eng.EvalStr('"hello again".upper()'));
      WriteLn('workdir: '+ eng.EvalStr('__import__("os").getcwd()'));
      writeln('syncheck '+
              botostr(eng.CheckEvalSyntax('print("powers:",[x**2 for x in range(10)])'))

      eng.ExecString('print("powers:",[x**2 for x in range(10)])');
      writeln('ExecSynCheck1 '+botostr(eng.CheckExecSyntax(PYCMD)));
      eng.ExecString(PYCMD);
      writeln('ExecSynCheck2 '+botostr(eng.CheckExecSyntax(myloadscript)));
      writeln('ExecSynCheck3 '+
            botostr(eng.CheckExecSyntax(filetostring(PYSCRIPT))));
      eng.ExecString(filetostring(PYSCRIPT));
      writeln(eng.Run_CommandAsString('print("powers:",[x**2 for x in range(10)])',eval_
      writeln(eng.Run_CommandAsString('sum([x for x in range(201)])',eval_input));
      pymemo.update;
    end
      else writeln('invalid library handle! '+Getlasterrortext);
    writeln('PythonOK: '+botostr(PythonOK));
  except
     eng.raiseError;
     writeln('PyErr '+ExceptionToString(ExceptionType, ExceptionParam));
  finally
     eng.free;
  end;
  out1.free;
  //pyImport(PyModule);
end;
```

The procedure raiseError helps to find errors for example:

Exception: : SRE module mismatch.
Make sure you do not have any mismatch between Python interpreter version used (like 3.7) and the 're' python module (like 3.6.1).

The resolution of Dlls has changed in Python 3.8 for Windows.

```
New in version 3.8: Previous versions of CPython would resolve Dlls using the default beh
```

And here's the reference output from *Procedure PYLaz_P4D_Demo2*:

```
DLLhandle: TRUE
evens: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196]
gauss: 5050
gauss2: 5050
mathstr: py py py py py py py
builtins: ['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException',
'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning',
'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError',
'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError',
'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit',
'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError',
'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt',
'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None',
'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError',
'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError',
'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError',
'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError',
'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True',
'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError',
'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning',
'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', 'build_class', 'debug',
'doc', 'import', 'loader', 'name', 'package', 'spec', 'abs', 'all', 'any', 'ascii',
'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile',
'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate',
'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr',
'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance',
'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview',
'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit',
'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted',
'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
upperstr: HELLO AGAIN
workdir: C:\maXbox\maxbox3\maxbox3\maXbox3
syncheck TRUE
ExecSynCheck1 TRUE
ExecSynCheck2 TRUE
ExecSynCheck3 TRUE
None
20100
PythonOK: TRUE
 mX4 executed: 28/07/2021 18:23:31 Runtime: 0:0:1.609 Memload: 43% use
```

Conclusion: The **eval()** method parses the expression passed to it and runs python expression(code) (but no statements) within the program. For you and for me 5 functions are crucial:

```
Function CheckEvalSyntax(const str: AnsiString):Boolean');
Function CheckExecSyntax(const str: AnsiString):Boolean');
Procedure ExecString(const command: AnsiString);');
Procedure ExecString3(const command: AnsiString);');//alias
Procedure ExecStrings4(strings: TStrings);');
Function EvalStringAsStr(const command: AnsiString):string');//alias
Function EvalStr(const command: AnsiString): string');
```

Also, consider the situation when you have imported **os** module in your python program like above *WriteLn('workdir: '+ eng.EvalStr('**import**("os").getcwd()'));*. The os module provides portable way to use operating system functionalities

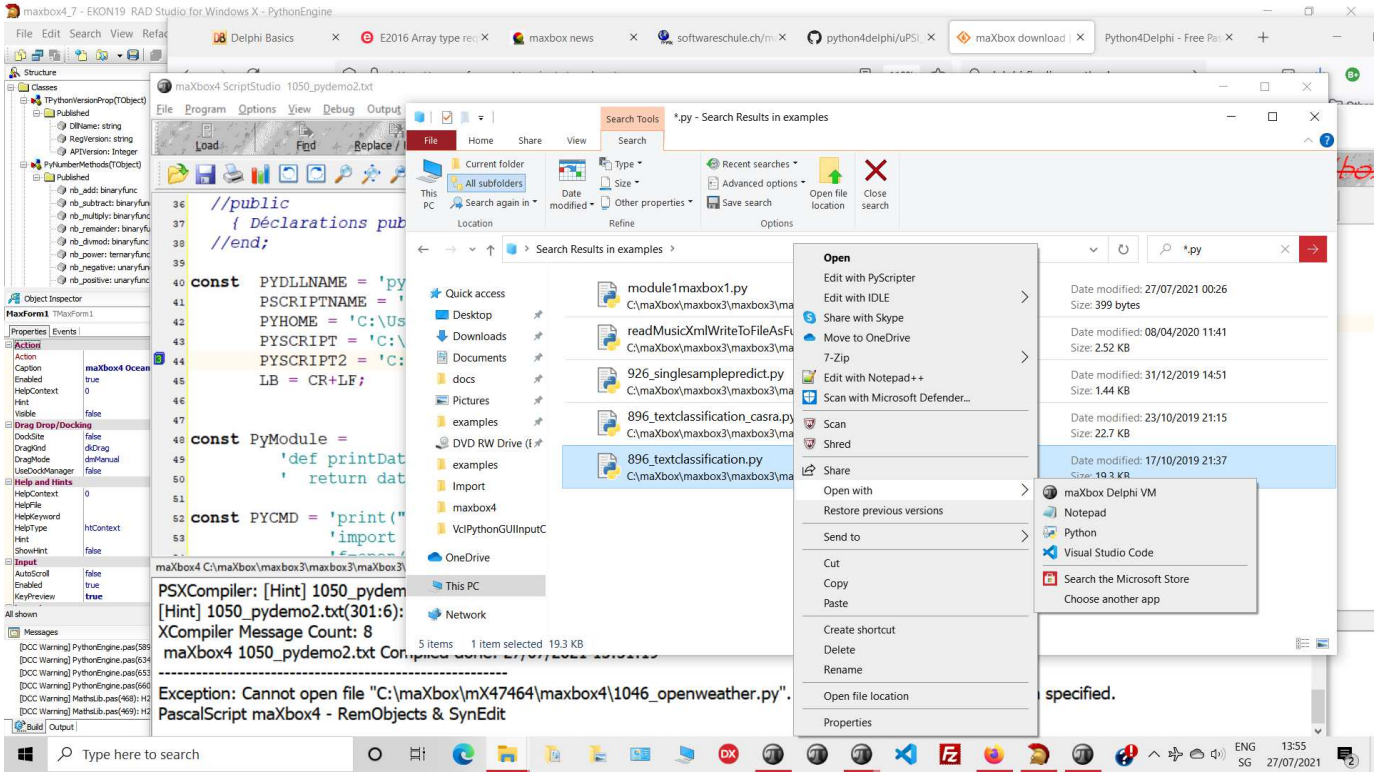
like: read or write a file. A single command can delete all files in your system!

So `eval` expects an expression, `import` is a statement. That said, what you can trying is the following combination:

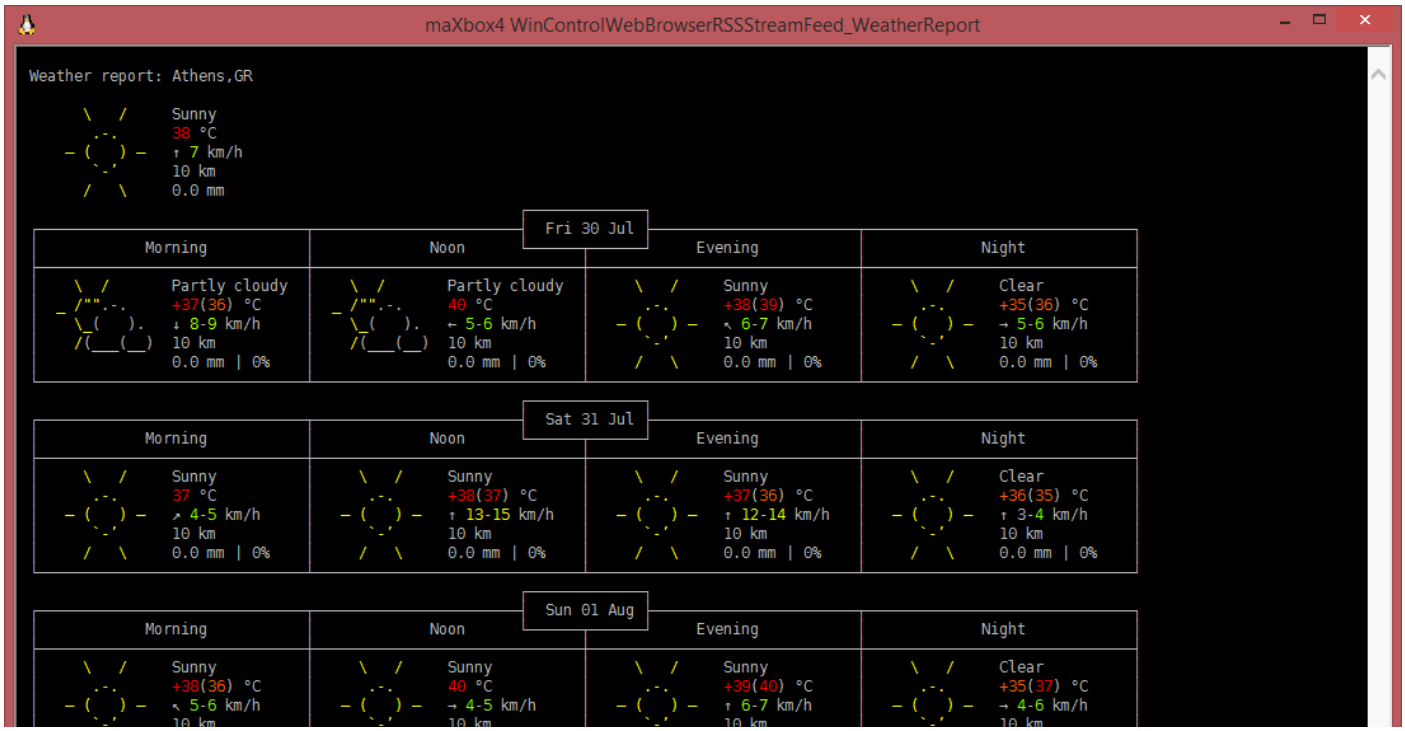
```
Println('exec as eval: '+eng.EvalStr('exec("import os as o")'));
Println('exec: '+eng.EvalStr('o.getcwd()'));
>>> exec as eval: None
>>> exec: C:\maXbox\mX47464\maxbox4
writeln('uuid: '+eng.evalstr('exec("import uuid") or str(uuid.uuid4())'));
>>> uuid: 3b2e10f9-0e31-4961-9246-00852fd508bd
```

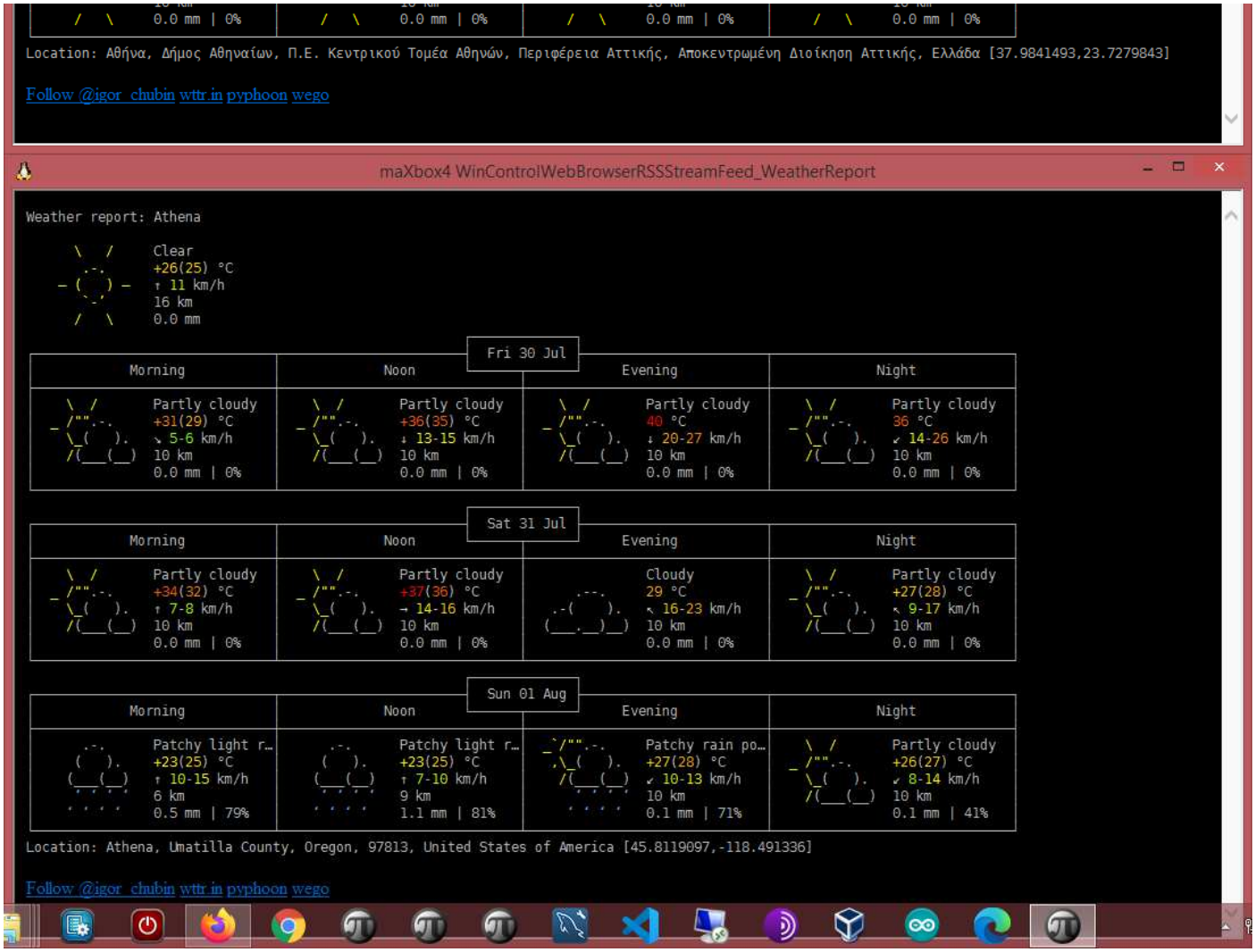
You can use `exec` in eval instead if you intend to import the module or also *ExecString*: it depends on the global or local namespace you set, means also the second line knows the import statement from first line:

```
eng.ExecString('import math');
Println('evalexec: '+eng.EvalStr('dir(math)'));
```



Compile the Pythonengine to run in maXbox and execute scripts in the code.





REST API of **https://wttr.in (https://wttr.in)** and PyOWM in combination out of maXbox
Posted in **maXbox**, **Python** Tagged **Coding**, **Git** **3 Comments**

3 thoughts on “Python4maXbox Code”

1. **maxbox4** says:
July 28, 2021 at 9:45 pm **Edit**

```
function CleanString(const s : AnsiString; AppendLF : Boolean) : AnsiString;
var
i : Integer;
begin
result := s;
if s = '' then
Exit;
i := Pos(AnsiString(CR),s);
while i > 0 do
begin
Delete( result, i, 1 );
i := PosEx(AnsiString(CR),result, i); //fix
end;
if AppendLF and (result[length(result)] LF) then
Result := Result + LF;
end;
```

REPLY

maxbox4 says:

July 30, 2021 at 3:07 pm **Edit**
>>> mystr
'hello world, how do i enter line breaks?'
>>> mystr.replace(' ', '')
'helloworld,howdoienterlinebreaks?'

In the example above, I replaced all spaces. The string '\n' represents newlines. And \r represents carriage returns (if you're on windows, you might be getting these and a second replace will handle them for you!). basically:

# you probably want to use a space ' ' to replace `\n`
mystring = mystring.replace('\n', ' ').replace('\r', '')

Note also, that it is a bad idea to call your variable string, as this shadows the module string. Another name I'd avoid but would love to use sometimes: file. For the same reason a semantic syntax checker.

REPLY

maxbox4 says:
July 30, 2021 at 3:13 pm **Edit**
Create a string containing line breaks
Newline code \n（LF）, \r\n（CR + LF）
Triple quote ''' or """
With indent
Concatenate a list of strings on new lines
Split a string into a list by line breaks: splitlines()
Remove or replace line breaks
Output with print() without a trailing newline