tocdepth: 2

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 3)

Unknown directive type "highlight".

```
.. highlight:: none
```

Python on Windows FAQ

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 11)

Unknown directive type "only".

```
.. only:: html

   .. contents::
```

How do I run a Python program under Windows?

This is not necessarily a straightforward question. If you are already familiar with running programs from the Windows command line then everything will seem obvious; otherwise, you might need a little more guidance.

Unless you use some sort of integrated development environment, you will end up *typing* Windows commands into what is variously referred to as a "DOS window" or "Command prompt window". Usually you can create such a window from your search bar by searching for `cmd`. You should be able to recognize when you have started such a window because you will see a Windows "command prompt", which usually looks like this:

```
C:\>
```

The letter may be different, and there might be other things after it, so you might just as easily see something like:

```
D:\YourName\Projects\Python>
```

depending on how your computer has been set up and what else you have recently done with it. Once you have started such a window, you are well on the way to running Python programs.

You need to realize that your Python scripts have to be processed by another program called the Python *interpreter*. The interpreter reads your script, compiles it into bytecodes, and then executes the bytecodes to run your program. So, how do you arrange for the interpreter to handle your Python?

First, you need to make sure that your command window recognises the word "py" as an instruction to start the interpreter. If you have opened a command window, you should try entering the command `py` and hitting return:

```
C:\Users\YourName> py
```

You should then see something like:

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

You have started the interpreter in "interactive mode". That means you can enter Python statements or expressions interactively and have them executed or evaluated while you wait. This is one of Python's strongest features. Check it by entering a few expressions of your choice and seeing the results:

```
>>> print("Hello")
Hello
>>> "Hello" * 3
'HelloHelloHello'
```

Many people use the interactive mode as a convenient yet highly programmable calculator. When you want to end your interactive Python session, call the :func:`exit` function or hold the :kbd:`Ctrl` key down while you enter a :kbd:`Z`, then hit the ":kbd:`Enter`" key to get back to your Windows command prompt.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 84); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 84); *backlink*

Unknown interpreted text role "kbd".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 84); *backlink*

Unknown interpreted text role "kbd".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 84); *backlink*

Unknown interpreted text role "kbd".

You may also find that you have a Start-menu entry such as :menuselection:`Start --> Programs --> Python 3.x --> Python (command line)` that results in you seeing the >>> prompt in a new window. If so, the window will disappear after you call the :func:`exit` function or enter the :kbd:`Ctrl-Z` character; Windows is running a single "python" command in the window, and closes it when you terminate the interpreter.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 90); *backlink*

Unknown interpreted text role "menuselection".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 90); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 90); *backlink*

Unknown interpreted text role "kbd".

Now that we know the `py` command is recognized, you can give your Python script to it. You'll have to give either an absolute or a relative path to the Python script. Let's say your Python script is located in your desktop and is named `hello.py`, and your command prompt is nicely opened in your home directory so you're seeing something similar to:

```
C:\Users\YourName>
```

So now you'll ask the `py` command to give your script to Python by typing `py` followed by your script path:

```
C:\Users\YourName> py Desktop\hello.py
hello
```

How do I make Python scripts executable?

On Windows, the standard Python installer already associates the .py extension with a file type (Python.File) and gives that file type an open command that runs the interpreter (`D:\Program Files\Python\python.exe "%1" %*`). This is enough to make scripts executable from the command prompt as 'foo.py'. If you'd rather be able to execute the script by simple typing 'foo' with no extension you need to add .py to the PATHEXT environment variable.

Why does Python sometimes take so long to start?

Usually Python starts very quickly on Windows, but occasionally there are bug reports that Python suddenly begins to take a long time to start up. This is made even more puzzling because Python will work fine on other Windows systems which appear to be configured identically.

The problem may be caused by a misconfiguration of virus checking software on the problem machine. Some virus scanners have been known to introduce startup overhead of two orders of magnitude when the scanner is configured to monitor all reads from the filesystem. Try checking the configuration of virus scanning software on your systems to ensure that they are indeed configured identically. McAfee, when configured to scan all file system read activity, is a particular offender.

How do I make an executable from a Python script?

See :ref:`faq-create-standalone-binary` for a list of tools that can be used to make executables.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 143); *backlink*

Unknown interpreted text role "ref".

Is a *.pyd file the same as a DLL?

Yes, .pyd files are dll's, but there are a few differences. If you have a DLL named `foo.pyd`, then it must have a function `PyInit_foo()`. You can then write Python "import foo", and Python will search for foo.pyd (as well as foo.py, foo.pyc) and if it finds it, will attempt to call `PyInit_foo()` to initialize it. You do not link your .exe with foo.lib, as that would cause Windows to require the DLL to be present.

Note that the search path for foo.pyd is PYTHONPATH, not the same as the path that Windows uses to search for foo.dll. Also, foo.pyd need not be present to run your program, whereas if you linked your program with a dll, the dll is required. Of course, foo.pyd is required if you want to say `import foo`. In a DLL, linkage is declared in the source code with `__declspec(dllexport)`. In a .pyd, linkage is defined in a list of available functions.

How can I embed Python into a Windows application?

Embedding the Python interpreter in a Windows app can be summarized as follows:

1. Do _not_ build Python into your .exe file directly. On Windows, Python must be a DLL to handle importing modules that are themselves DLL's. (This is the first key undocumented fact.) Instead, link to :file:`python{NN}.dll`; it is typically installed in `C:\Windows\System`. *NN* is the Python version, a number such as "33" for Python 3.3.

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 170); *backlink*

   Unknown interpreted text role "file".

   You can link to Python in two different ways. Load-time linking means linking against :file:`python{NN}.lib`, while run-time linking means linking against :file:`python{NN}.dll`. (General note: :file:`python{NN}.lib` is the so-called "import lib" corresponding to :file:`python{NN}.dll`. It merely defines symbols for the linker.)

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 176); *backlink*

   Unknown interpreted text role "file".

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 176); *backlink*

   Unknown interpreted text role "file".

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 176); *backlink*

   Unknown interpreted text role "file".

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 176); *backlink*

   Unknown interpreted text role "file".

   Run-time linking greatly simplifies link options; everything happens at run time. Your code must load :file:`python{NN}.dll` using the Windows `LoadLibraryEx()` routine. The code must also use access routines and data in :file:`python{NN}.dll` (that is, Python's C API's) using pointers obtained by the Windows `GetProcAddress()` routine. Macros can make using these pointers transparent to any C code that calls routines in Python's C API.

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 182);

backlink

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst`, **line 182**); *backlink*

Unknown interpreted text role "file".

Borland note: convert :file:`python{NN}.lib` to OMF format using Coff2Omf.exe first.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst`, **line 189**); *backlink*

Unknown interpreted text role "file".

2. If you use SWIG, it is easy to create a Python "extension module" that will make the app's data and methods available to Python. SWIG will handle just about all the grungy details for you. The result is C code that you link *into* your .exe file (!) You do _not_ have to create a DLL file, and this also simplifies linking.

3. SWIG will create an init function (a C function) whose name depends on the name of the extension module. For example, if the name of the module is leo, the init function will be called initleo(). If you use SWIG shadow classes, as you should, the init function will be called initleoc(). This initializes a mostly hidden helper class used by the shadow class.

   The reason you can link the C code in step 2 into your .exe file is that calling the initialization function is equivalent to importing the module into Python! (This is the second key undocumented fact.)

4. In short, you can use the following code to initialize the Python interpreter with your extension module.

   ```
   #include <Python.h>
   ...
   Py_Initialize();  // Initialize Python.
   initmyAppc();  // Initialize (import) the helper class.
   PyRun_SimpleString("import myApp");  // Import the shadow class.
   ```

5. There are two problems with Python's C API which will become apparent if you use a compiler other than MSVC, the compiler used to build pythonNN.dll.

   Problem 1: The so-called "Very High Level" functions that take FILE * arguments will not work in a multi-compiler environment because each compiler's notion of a struct FILE will be different. From an implementation standpoint these are very _low_ level functions.

   Problem 2: SWIG generates the following code when generating wrappers to void functions:

   ```
   Py_INCREF(Py_None);
   _resultobj = Py_None;
   return _resultobj;
   ```

   Alas, Py_None is a macro that expands to a reference to a complex data structure called _Py_NoneStruct inside pythonNN.dll. Again, this code will fail in a mult-compiler environment. Replace such code by:

   ```
   return Py_BuildValue("");
   ```

   It may be possible to use SWIG's `%typemap` command to make the change automatically, though I have not been able to get this to work (I'm a complete SWIG newbie).

6. Using a Python shell script to put up a Python interpreter window from inside your Windows app is not a good idea; the resulting window will be independent of your app's windowing system. Rather, you (or the wxPythonWindow class) should create a "native" interpreter window. It is easy to connect that window to the Python interpreter. You can redirect Python's i/o to _any_ object that supports read and write, so all you need is a Python object (defined in your extension module) that contains read() and write() methods.

How do I keep editors from inserting tabs into my Python source?

The FAQ does not recommend using tabs, and the Python style guide, PEP 8, recommends 4 spaces for distributed Python code; this is also the Emacs python-mode default.

Under any editor, mixing tabs and spaces is a bad idea. MSVC is no different in this respect, and is easily configured to use spaces: Take :menuselection:`Tools --> Options --> Tabs`, and for file type "Default" set "Tab size" and "Indent size" to 4, and select the "Insert spaces" radio button.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 265); *backlink*

Unknown interpreted text role "menuselection".

Python raises :exc:`IndentationError` or :exc:`TabError` if mixed tabs and spaces are causing problems in leading whitespace. You may also run the :mod:`tabnanny` module to check a directory tree in batch mode.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 270); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 270); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 270); *backlink*

Unknown interpreted text role "mod".

How do I check for a keypress without blocking?

Use the :mod:`msvcrt` module. This is a standard Windows-specific extension module. It defines a function `kbhit()` which checks whether a keyboard hit is present, and `getch()` which gets one character without echoing it.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\faq\[cpython-main][Doc][faq]windows.rst, line 279); *backlink*

Unknown interpreted text role "mod".