Working Example 1

Function

Accessing Cython functions from a Python file

In this example, some functions defined in a Cython file, are wrapped and accessed from a Python file.

For getting results, simply run the ***"script.sh"*** file (it runs ***"setup.py"*** file and then ***"test.py"*** file).

The Cython file ***"pyMultiply.pyx"*** defines two functions, named ***"cpSum"*** and ***"cMultiply"***. These functions respectively return the addition and multiplication of given two numbers. The function ***"cpSum"*** can be called from both Python and C/C++, because it is defined with a keyword ***"cpdef"***, whereas the function ***"cMultiply"*** could only be called from C/C++, because it is defined with ***"cdef"***. After that, wrappers of these functions are defined with names ***"pyTestSum"*** and ***"pyTestMultiply"***.

This Cython file ***"pyMultiply.pyx"*** is used in ***"setup.py"*** to actually create a wrapper file (***.so*** file).

The Python file ***"test.py"*** imports the wrapper, defines two variables, passes them to the functions and gets result. Since the function ***"cpSum"*** is defined with ***"cpdef"***, therefore it can be called directly from Python as well as using its wrapper function ***"pyTestSum"***. But the function ***"cMultiply"*** cannot be called directly from Python because it is defined with ***"cdef"***. Hence, it can only be called using its wrapper function ***"pyTestMultiply"***.

Class

Accessing a C++ class from a Python file

In this example, a class in a C++ file is defined, which is wrapped into Cython and then accessed from a Python file.

For getting results, simply run the ***"script.sh"*** file (it runs ***"setup.py"*** file and then ***"test.py"*** file).

The file ***"class_example.h"*** defines a namespace ***"vehicles"***, in which there is a class ***"Car"*** that has variables and functions. The constructor, destructor and function are implemented in ***"class_example.cpp"*** file. This class is wrapped into Cython file ***"pyCar.pyx"***. The ***"setup.py"*** file performs the wrapping operation and generates a ***.so*** file which is imported in ***"test.py"*** file. So, using Python code, we can access C++ class. An object of C++ class is defined in Python and its functions are accessed.

There are two ways to create a wrapper of C++ file using Cython. In both methods, running ***"setup.py"*** file creates ***.so*** file that is imported into ***"test.py"*** file.

- In the method-1, in ***"setup.py"*** file, "***class_example.cpp"*** file is used along with "***pyCar.pyx"*** file. It automatically creates the object file from ***.cpp*** file, links with ***.pyx*** file and creates a ***.so*** file. So, it is not needed to manually create ***.o*** file from ***.cpp*** file, then create ***.a*** file from ***.o*** file and then link with ***.pyx*** file.

- In method-2, some commands have been run in ***"script.sh"*** to run ***"class_example.cpp"*** file and generate ***.o*** file from which ***.a*** file is generated. This ***.a*** file is then used with ***pyCar.pyx*** file in ***"setup.py"*** to create a ***.so*** file.

Working Example 2

Accessing a C function from a Python file

In this example, a function defined in a C file is wrapped into Cython and then accessed from a Python file.

For getting results, simply run the ***"script.sh"*** file (it runs ***"setup.py"*** file and then ***"main.py"*** file).

The ***"helloWorld.h"*** file contains a function ***"functionHelloWorld"***, which is implemented in ***"helloWorld.c"***. The Cython file ***"helloWorldCython.pyx"*** wraps this function. Running ***"setup.py"*** creates a ***.so*** file that is imported into ***"main.py"*** file. The ***"main.py"*** file accesses the function in C and passes it a string.

There are two ways to create a wrapper. In both the cases, a ***.so*** file is generated that is imported into python file. However, in Method-1, ***"setup.py"*** itself compiles the ***"helloWorld.c"***, creates its object and links with ***"helloWorldCython.pyx"***. Whereas in Method-2, two additional commands are written in ***"script.sh"*** file to create ***.o*** file from ***.c*** file and then generate ***.a*** file from ***.o*** file. This ***.a*** file is used in ***"setup.py"*** file along with ***"helloWorldCython.pyx"*** to create a ***.so*** file.

Working Example 3

Accessing a C++ class from a Python file

In this example, a class in a C++ file is defined, which is wrapped into Cython and then accessed from a Python file.

For getting results, simply run the ***"script.sh"*** file (it runs ***"setup.py"*** file and then ***"test.py"*** file).

The file ***"Rectangle.h"*** defines a namespace ***"shapes"***, in which there is a class ***"Rectangle"*** that has variables and functions. The constructor, destructor and functions are implemented in ***"Rectangle.cpp"*** file. This class is wrapped into a Cython file ***"rect.pyx"*** . The ***"Rectangle.pxd"*** works as a header file of Cython ***.pyx*** file. The ***"setup.py"*** file performs the wrapping operation and generates a ***.so*** file which is imported in ***"test.py"*** file. So, using Python code, we can access C++ class. An object of C++ class is defined in Python and its functions are accessed.

There are two ways to create a wrapper of C++ file using Cython. In both methods, running ***"setup.py"*** file creates ***.so*** file that is imported into ***"test.py"*** file.

- In the method-1, in ***"setup.py"*** file, "***Rectangle.cpp"*** file is used along with "***rect.pyx"*** file. It automatically creates the object file from ***.cpp*** file, links with ***.pyx*** file and creates a ***.so*** file.

So, it is not needed to manually create ***.o*** file from ***.cpp*** file, then create ***.a*** file from ***.o*** file and then link with ***.pyx*** file.

- In method-2, some commands have been run in ***"script.sh"*** to run ***"class_example.cpp"*** file and generate ***.o*** file from which ***.a*** file is generated. This ***.a*** file is then used with "***rect.pyx"*** file in ***"setup.py"*** to create a ***.so*** file.

Working Example 4

Function

Accessing C++ functions from a Python file

In this example, two functions in a C++ file are defined, which are wrapped into Cython and then accessed from a Python file.

For getting results, simply run the ***"script.sh"*** file (it runs ***"setup.py"*** file and then ***"test.py"*** file).

The ***"ExmpAddMult.h"*** defines two functions which are implemented in ***"ExmpAddMult.cpp"***. Both functions are wrapped into Cython in ***"ExmpAddMultCy.pyx"*** file. This Cython file creates a wrapper and generates a ***.so*** file, which is imported into the Python file ***"test.py"***.

There are two ways to create a wrapper of C++ file using Cython. In both methods, running ***"setup.py"*** file creates ***.so*** file that is imported into ***"test.py"*** file.

- In the method-1, in ***"setup.py"*** file, "***ExmpAddMult.cpp"*** file is used along with "***ExmpAddMultCy.pyx"*** file. It automatically creates the object file from ***.cpp*** file, links with ***.pyx*** file and creates a ***.so*** file. So, it is not needed to manually create ***.o*** file from ***.cpp*** file, then create ***.a*** file from ***.o*** file and then link with ***.pyx*** file.
- In method-2, some commands have been run in ***"script.sh"*** to run ***"ExmpAddMult.cpp"*** file and generate ***.o*** file from which ***.a*** file is generated. This ***.a*** file is then used with "***ExmpAddMultCy.pyx"*** file in ***"setup.py"*** to create a ***.so*** file.

Class

Accessing a C++ class from a Python file

It is exactly same as above ***"Function"*** except the difference that C++ functions belong to a class, which is defined in a namespace. So, Cython file contains wrapper of a class, instead of wrappers of functions. It has also two ways of linking Cython file with C++ file.