**Can you check what is wrong wi...**

New chat

**2 December 2024 at 15:38**

**You said:**

Can you check what is wrong with this UDF:import rdrand rng = rdrand.RdRandom() @xw.func def generate\_random\_numbers\_rdrand(num): # Generate the random numbers random\_numbers = [[int(rng.random() % (10\*\*15 - 10\*\*14)) + 10\*\*14] for \_ in range(int(num))] return random\_numbersIt returns entire range in excel like this:100000000000000 100000000000000 100000000000000 100000000000000 100000000000000 100000000000000

**Copilot said:**

It looks like the issue with your UDF (User Defined Function) is that the random numbers generated are not varying as expected. This is likely due to the way the random numbers are being calculated. The modulo operation (%) with a large number like (10^{15} - 10^{14}) might not be producing the desired randomness.

Here's a revised version of your UDF that should generate more varied random numbers:

import rdrand

import xlwings as xw

rng = rdrand.RdRandom()

@xw.func

def generate\_random\_numbers\_rdrand(num):

# Generate the random numbers

random\_numbers = [[rng.randint(10\*\*14, 10\*\*15 - 1)] for \_ in range(int(num))]

return random\_numbers

In this version, I used rng.randint(10\*\*14, 10\*\*15 - 1) to generate random integers within the desired range. This should give you a better distribution of random numbers.

Give this a try and let me know if it resolves the issue!

Edit in Pages

Copy

1 of 30 responsesAI-generated content may be incorrect

**2 December 2024 at 15:56**

**You said:**

Okay that was a success. Now Let's work on following excel formula:=SUM(MAP(D3:D17,J3:J3,E3#,LAMBDA(x,y,z,x\*((1+(y/4))^z))))As you can see, the second argument in MAP is a single cell. Can we emulate single cell as count of argument previous to it? D3:D17At present this formula is failing.

**Copilot said:**

Great to hear that the UDF worked! For your Excel formula, you want to use a single cell (J3) as a constant across the range D3:D17. To achieve this, you can use the LET function to define the constant and then use it within the MAP function.

Here's how you can modify your formula:

=LET(

y, J3,

SUM(MAP(D3:D17, E3#, LAMBDA(x, z, x \* ((1 + (y / 4)) ^ z))))

)

In this formula:

* LET is used to define y as the value in cell J3.
* MAP then uses y as a constant for each element in the range D3:D17.

This should work as intended, applying the value of J3 across the entire range. Give it a try and let me know if it resolves the issue!

Edit in Pages

Copy

2 of 30 responsesAI-generated content may be incorrect

**2 December 2024 at 16:35**

**You said:**

What is wrong with following UDF. Why Can't I see the function in excel in spite of repeated efforts to restart xlwings server and importing UDFs:from cffi import FFI import xlwings as xw ffi = FFI() #NUM\_NUMBERS = 100000 #NUM\_THREADS = 16 # Define the functions in the DLL ffi.cdef(""" int rdrand64\_step(unsigned long long \*rand); void generate\_random\_numbers(int num\_threads, int num\_numbers); unsigned long long\* get\_numbers(); void free\_numbers(unsigned long long \*numbers); """) # Load the DLL #C = ffi.dlopen('D:\\OneDrive - 0yt2k\\Compiled dlls & executables\\rdrand\_multithreaded\_new\_ucrt\_gcc.dll') C = ffi.dlopen('D:\\dev\\dll\\rdrand\_multithreaded\_new.dll') @xw.func def generate\_and\_get\_data(NUM\_THREADS, NUM\_NUMBERS): NUM\_THREADS = int(NUM\_THREADS) NUM\_NUMBERS = int(NUM\_NUMBERS) C.generate\_random\_numbers(NUM\_THREADS, NUM\_NUMBERS) numbers\_ptr = C.get\_numbers() numbers = [[int(numbers\_ptr[i])] for i in range(NUM\_NUMBERS)] C.free\_numbers(numbers\_ptr) return numbers

**Copilot said:**

There are a few potential reasons why your UDF might not be appearing in Excel. Let's go through some common issues and solutions:

1. **Function Registration**: Ensure that the function is properly registered with xlwings. Sometimes, the function might not be recognized if it's not correctly registered.
2. **Excel Add-in**: Make sure the xlwings add-in is installed and enabled in Excel. You can check this in Excel under File > Options > Add-ins.
3. **Module Import**: Verify that the module containing your UDF is correctly imported in the xlwings add-in settings. You can do this by opening the xlwings add-in settings and ensuring that the module name is listed under the "UDF Modules" section.
4. **Restart Excel**: Sometimes, simply restarting Excel can help in recognizing new or updated UDFs.
5. **Error Handling**: Ensure there are no errors in your code that might prevent the function from being loaded. You can add some print statements or logging to check if the function is being executed.
6. **Correct Path**: Double-check the path to your DLL to ensure it's correct and accessible.

Here's a checklist to help you troubleshoot:

* **Check Function Registration**:
* import xlwings as xw
* @xw.func
* def my\_test\_function():
* return "Hello, Excel!"
* **Verify Add-in Installation**:
  + Go to File > Options > Add-ins in Excel.
  + Ensure xlwings is listed and enabled.
* **Check UDF Module Import**:
  + Open the xlwings add-in settings.
  + Ensure your module is listed under "UDF Modules".
* **Restart Excel**:
  + Close and reopen Excel to refresh the UDFs.
* **Add Logging**:
* import logging
* logging.basicConfig(level=logging.DEBUG)
* @xw.func
* def generate\_and\_get\_data(NUM\_THREADS, NUM\_NUMBERS):
* logging.debug("Function called with NUM\_THREADS=%d, NUM\_NUMBERS=%d", NUM\_THREADS, NUM\_NUMBERS)
* # Your existing code here

If you've gone through these steps and the function still doesn't appear, there might be an issue with the DLL or the way it's being loaded. Ensure that the DLL is correctly compiled and accessible from the path specified.

Let me know if any of these steps help or if you need further assistance!

Edit in Pages

Copy

3 of 30 responsesAI-generated content may be incorrect

**2 December 2024 at 16:39**

**You said:**

Following the source code of above dll. How to compile it in Msys using gcc / g++ - I have already installed tool chains:#include <pthread.h> #include <immintrin.h> #include <stdint.h> #include <stdio.h> #include <stdlib.h> #define MAX\_DIGITS 15 typedef struct { uint64\_t \*numbers; int start; int end; } thread\_data\_t; unsigned long long \*numbers; // Make numbers global so it can be accessed by get\_numbers \_\_declspec(dllexport) int rdrand64\_step(unsigned long long \*rand) { return \_rdrand64\_step(rand); } void \*generate\_random\_numbers\_thread(void \*arg) { thread\_data\_t \*data = (thread\_data\_t \*)arg; unsigned long long rand; for (int i = data->start; i < data->end; i++) { do { if (!rdrand64\_step(&rand)) { printf("Failed to generate a random number.\n"); } rand = rand % (999999999999999 + 1); } while (rand < 100000000000000); // Ensure the number is 15 digits long data->numbers[i] = rand; } return NULL; } \_\_declspec(dllexport) void generate\_random\_numbers(int num\_threads, int num\_numbers) { pthread\_t threads[num\_threads]; thread\_data\_t thread\_data[num\_threads]; numbers = (unsigned long long\*)malloc(num\_numbers \* sizeof(unsigned long long)); if (numbers == NULL) { fprintf(stderr, "Failed to allocate memory.\n"); return; } int numbers\_per\_thread = num\_numbers / num\_threads; for (int i = 0; i < num\_threads; i++) { thread\_data[i].numbers = numbers; thread\_data[i].start = i \* numbers\_per\_thread; thread\_data[i].end = (i == num\_threads - 1) ? num\_numbers : (i + 1) \* numbers\_per\_thread; if (pthread\_create(&threads[i], NULL, generate\_random\_numbers\_thread, &thread\_data[i]) != 0) { fprintf(stderr, "Failed to create thread.\n"); return; } } for (int i = 0; i < num\_threads; i++) { if (pthread\_join(threads[i], NULL) != 0) { fprintf(stderr, "Failed to join thread.\n"); return; } } // numbers now contains your random numbers // don't forget to free(numbers) when you're done with it } \_\_declspec(dllexport) unsigned long long\* get\_numbers() { return numbers; } \_\_declspec(dllexport) void free\_numbers(unsigned long long \*numbers) { free(numbers); }

**Copilot said:**

To compile your C code using GCC in MSYS2, follow these steps:

1. **Open MSYS2 Terminal**: Launch the MSYS2 terminal from your Start menu.
2. **Navigate to Your Source Code Directory**: Use the cd command to navigate to the directory where your source code is located. For example:
3. cd /path/to/your/source/code
4. **Compile the Code**: Use the gcc command to compile your code. Since your code uses pthreads, you need to link the pthread library. Here's the command:
5. gcc -o generate\_random\_numbers.dll -shared -fPIC -pthread your\_source\_code.c
   * -o generate\_random\_numbers.dll: Specifies the output file name.
   * -shared: Creates a shared library (DLL).
   * -fPIC: Generates position-independent code (necessary for shared libraries).
   * -pthread: Links the pthread library.
6. **Verify the DLL**: After compilation, you should see generate\_random\_numbers.dll in your directory. You can use it in your Python code as intended.

Here's a step-by-step example assuming your source code is saved in a file named rdrand\_multithreaded.c:

cd /path/to/your/source/code

gcc -o rdrand\_multithreaded.dll -shared -fPIC -pthread rdrand\_multithreaded.c

This should compile your code into a DLL that you can use with your Python script.

If you encounter any errors during compilation, please share the error messages, and I can help troubleshoot further!

Edit in Pages

Copy

4 of 30 responsesAI-generated content may be incorrect

**2 December 2024 at 16:40**

**You said:**

Have not skipped mrdrand?

**Copilot said:**

It looks like I missed addressing the mrdrand function in your code. Let's include it in the compilation process. Here's how you can compile your code with the mrdrand function included:

1. **Ensure mrdrand is Defined**: Make sure the mrdrand function is defined in your source code. If it's part of another file, include that file in the compilation.
2. **Compile with mrdrand**: If mrdrand is part of the same file, the previous compilation command should work. If it's in a separate file, you need to compile both files together.

Assuming mrdrand is in a separate file named mrdrand.c, you can compile both files together like this:

gcc -o rdrand\_multithreaded.dll -shared -fPIC -pthread rdrand\_multithreaded.c mrdrand.c

If mrdrand is in the same file, the previous command should suffice:

gcc -o rdrand\_multithreaded.dll -shared -fPIC -pthread rdrand\_multithreaded.c

Make sure all necessary source files are included in the compilation command. If you encounter any specific errors or need further assistance, feel free to share the details!

Edit in Pages

Copy

5 of 30 responsesAI-generated content may be incorrect

**2 December 2024 at 16:41**

**You said:**

But I am referring to switch -mrdrnd or something like that

**Copilot said:**

It seems like you might be referring to a specific compiler switch for GCC related to random number generation. However, there isn't a standard -mrdrnd switch in GCC. Instead, you can use the -mrdrnd option to enable the use of the RDRAND instruction on x86 processors that support it.

Here's how you can compile your code with the -mrdrnd switch:

1. **Open MSYS2 Terminal**: Launch the MSYS2 terminal from your Start menu.
2. **Navigate to Your Source Code Directory**: Use the cd command to navigate to the directory where your source code is located. For example:
3. cd /path/to/your/source/code
4. **Compile the Code with -mrdrnd**: Use the gcc command to compile your code with the -mrdrnd switch. Here's the command:
5. gcc -o rdrand\_multithreaded.dll -shared -fPIC -pthread -mrdrnd rdrand\_multithreaded.c
   * -o rdrand\_multithreaded.dll: Specifies the output file name.
   * -shared: Creates a shared library (DLL).
   * -fPIC: Generates position-independent code (necessary for shared libraries).
   * -pthread: Links the pthread library.
   * -mrdrnd: Enables the use of the RDRAND instruction.

This should compile your code into a DLL that you can use with your Python script. If you encounter any errors during compilation, please share the error messages, and I can help troubleshoot further!

Edit in Pages

Copy

6 of 30 responsesAI-generated content may be incorrect

**2 December 2024 at 16:49**

**You said:**

I recompiled and tried in Python directly. But get this error:>>> C = ffi.dlopen('D:\\dev\\dll\\rdrand\_multithreaded\_new.dll') Traceback (most recent call last): File "<stdin>", line 1, in <module> File "D:\Programs\Python\Lib\site-packages\cffi\api.py", line 150, in dlopen lib, function\_cache = \_make\_ffi\_library(self, name, flags) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "D:\Programs\Python\Lib\site-packages\cffi\api.py", line 834, in \_make\_ffi\_library backendlib = \_load\_backend\_lib(backend, libname, flags) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "D:\Programs\Python\Lib\site-packages\cffi\api.py", line 830, in \_load\_backend\_lib return backend.load\_library(path, flags) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ OSError: cannot load library 'D:\dev\dll\rdrand\_multithreaded\_new.dll': error 0x7eI have defined os path as: >>> os.environ['PATH'] = r'D:\Programs\Msys2\ucrt64\bin;D:\Programs\Msys2\ucrt64\lib;D:\Programs\Msys2\ucrt64\include;D:\dev\dll;D:\bSo all backend libraries are covered, I think.

**Copilot said:**

The error code 0x7e typically indicates that a required DLL is missing or cannot be found. Here are a few steps to troubleshoot and resolve this issue:

1. **Check DLL Dependencies**:
   * Use a tool like Dependency Walker to check if your DLL (rdrand\_multithreaded\_new.dll) has any missing dependencies. This tool will show you all the dependent DLLs and highlight any that are missing.
2. **Ensure Correct DLL Path**:
   * Double-check that the path to your DLL is correct and that the DLL is indeed located at D:\dev\dll\rdrand\_multithreaded\_new.dll.
3. **Add DLL Directory to PATH**:
   * Ensure that the directory containing your DLL is added to the PATH environment variable. You mentioned setting the PATH, but make sure it's correctly set in your Python environment:
   * import os
   * os.environ['PATH'] = r'D:\Programs\Msys2\ucrt64\bin;D:\Programs\Msys2\ucrt64\lib;D:\Programs\Msys2\ucrt64\include;D:\dev\dll;' + os.environ['PATH']
4. **Check for Missing System Libraries**:
   * Sometimes, the issue can be due to missing system libraries. Ensure that all required system libraries (like ucrtbase.dll for Universal C Runtime) are present in your system or the specified paths.
5. **Use os.add\_dll\_directory**:
   * For Python 3.8 and later, you can use os.add\_dll\_directory to add the directory containing your DLL:
   * import os
   * os.add\_dll\_directory(r'D:\dev\dll')
6. **Recompile with Static Linking**:
   * If the issue persists, consider recompiling your DLL with static linking for all dependencies. This can help avoid runtime issues with missing DLLs.

Here's an example of how you might adjust your Python code to ensure the DLL path is correctly set:

import os

from cffi import FFI

# Add the DLL directory to the PATH

os.add\_dll\_directory(r'D:\dev\dll')

ffi = FFI()

ffi.cdef("""

int rdrand64\_step(unsigned long long \*rand);

void generate\_random\_numbers(int num\_threads, int num\_numbers);

unsigned long long\* get\_numbers();

void free\_numbers(unsigned long long \*numbers);

""")

# Load the DLL

C = ffi.dlopen('D:\\dev\\dll\\rdrand\_multithreaded\_new.dll')

Try these steps and let me know if you encounter any further issues!

Edit in Pages

Copy

7 of 30 responsesAI-generated content may be incorrect

**2 December 2024 at 16:52**

**You said:**

But all my msys libraries will not be covered in above case. They are under: D:\Programs\Msys2\ucrt64 - like bin, lib, D:\Programs\Msys2\ucrt64\x86\_64-w64-mingw32\bin, D:\Programs\Msys2\ucrt64\x86\_64-w64-mingw32\include, etc. Which ones I have to add?

**Copilot said:**

To ensure all necessary MSYS2 libraries are included, you should add the following directories to your PATH environment variable:

1. **ucrt64/bin**: This directory contains the main executables and DLLs.
2. **ucrt64/lib**: This directory contains the library files.
3. **ucrt64/include**: This directory contains the header files.
4. **ucrt64/x86\_64-w64-mingw32/bin**: This directory contains additional executables and DLLs specific to the MinGW-w64 toolchain.
5. **ucrt64/x86\_64-w64-mingw32/include**: This directory contains additional header files specific to the MinGW-w64 toolchain.

Here's how you can set the PATH environment variable in Python to include these directories:

import os

# Add MSYS2 directories to PATH

os.environ['PATH'] = (

r'D:\Programs\Msys2\ucrt64\bin;' +

r'D:\Programs\Msys2\ucrt64\lib;' +

r'D:\Programs\Msys2\ucrt64\include;' +

r'D:\Programs\Msys2\ucrt64\x86\_64-w64-mingw32\bin;' +

r'D:\Programs\Msys2\ucrt64\x86\_64-w64-mingw32\include;' +

os.environ['PATH']

)

from cffi import FFI

ffi = FFI()

ffi.cdef("""

int rdrand64\_step(unsigned long long \*rand);

void generate\_random\_numbers(int num\_threads, int num\_numbers);

unsigned long long\* get\_numbers();

void free\_numbers(unsigned long long \*numbers);

""")

# Load the DLL

C = ffi.dlopen('D:\\dev\\dll\\rdrand\_multithreaded\_new.dll')

By adding these directories to your PATH, you ensure that all necessary libraries and executables are available for your DLL to function correctly. If you still encounter issues, using a tool like Dependency Walker can help identify any missing dependencies

1

2

.

Give this a try and let me know if it resolves the issue!

1Environments - MSYS22MSYS2 - pkg-configwhich msys libraries to add to PATH for DLL

Edit in Pages

Copy

8 of 30 responsesAI-generated content may be incorrect