User

In the following C source file, can you modify the code to generate random numbers as integers instead of floats? It is actually a source file to be compiled into python package rdrand.

Bing

Sure, I can help with that. However, I would need to see the existing code first. Could you please provide the C source file or the relevant code snippet that you want to modify? This will help me understand the context and provide a more accurate solution.

User

#define PY_SSIZE_T_CLEAN #include <Python.h> #include <stdint.h>

// You may have some issues compiling this to 16 bit. Like getting python to work. #ifdef _LP64 #define IS64BIT 1 #else #define IS32BIT 1 #endif

// Willing to support other compilers, if you can get me access to them, and they // build python #ifdef **GNUC** #define USING_GCC 1 #elif **clang** #define USING_CLANG 1 #else #error Only support for gcc or clang currently #error if you port to another compiler, please #error send back the patch to https://github.com/stillson/rdrand #endif

#if PY_MAJOR_VERSION == 2 #define PYTHON2 1 #elif PY_MAJOR_VERSION == 3 #define PYTHON3 1 #else #error requires python 2 or 3 #endif

uint64_t get_bits_using_rdrand(void); uint64_t get_bits_using_rdseed(void); int RdRand_cpuid(void); int RdSeed_cpuid(void);

PyDoc_STRVAR(module_doc, “rdrand: Python interface to Intel and AMD hardware RNG\n”);

Bing

The code you’ve provided seems to be a part of the header file or the beginning of the C source file. It includes preprocessor directives, function declarations, and documentation, but it doesn’t contain the function definitions where the random numbers are generated.

To generate random integers instead of floats, you would typically need to modify the function where the random number is generated. If the function is currently generating a float between 0 and 1, you could multiply by the maximum integer you want and then cast to an integer.

However, without the rest of the code, especially the function definitions of `get_bits_using_rdrand` and `get_bits_using_rdseed`, it’s hard to provide a precise solution. Could you please provide the rest of the code? This will help me give you a more accurate answer.

User

For the python package which builds on the top of C files, where is that what you have described above: However, without the rest of the code, especially the function definitions of get_bits_using_rdrand and get_bits_using_rdseed, it’s hard to provide a precise solution. Could you please provide the rest of the code? This will help me give you a more accurate answer.

Bing

The functions `get_bits_using_rdrand` and `get_bits_using_rdseed` are likely defined in the C source files of the `rdrand` Python package. These files are typically included when you download the source code of the package.

The source code of the `rdrand` package can be found on its GitHub repository. Once you’ve downloaded the source code, you can look for the C source files (usually with the `.c` extension). The function definitions should be in one of these files.

If you’re unable to find the source code or have trouble understanding it, I’d be happy to help further if you can provide more details or specific code snippets.

User

Here is:

uint64_t get_bits_using_rdrand(void) { unsigned long int rando = 0; GETRAND(rando); return rando; }

and

get_bits_using_rdseed(void) { unsigned long int rando = 0; GETSEED(rando); return rando; }

But I think that part refers to some subfunctions. Can you change return part to convert to integers?

Bing

The functions `get_bits_using_rdrand` and `get_bits_using_rdseed` are already returning integers (`uint64_t` is an unsigned 64-bit integer type). The `unsigned long int rando = 0;` line is declaring `rando` as an unsigned long integer, and the `GETRAND(rando);` and `GETSEED(rando);` lines are presumably setting `rando` to a random value.

If you want to limit the range of the random integers, you can use the modulo operator (`%`). For example, if you want to generate random integers between 0 and 99, you can modify the return statement like this:

```
return rando % 100;
```

This will return the remainder of the division of `rando` by 100, which is always in the range 0 to 99. You can replace 100 with any other number to change the range. Note that this may introduce some bias if the range does not evenly divide the maximum value that `rando` can take. If this is a concern, there are more complex methods to generate unbiased random integers in a range. Let me know if you need help with that.

User

Okay - Following is my python file from that same source on github:

from random import Random as StdRandom from _rdrand import *

if HAS_RAND == 0: print(“This module requires a cpu which supports the” + " RdRand instruction") raise SystemError

if HAS_SEED == 0: rdseed_get_bits = None rdseed_get_bytes = None

class BaseRandom(StdRandom): “”“Base class for alternate random number generator using Intel’s RdRand or RdSeed instructions to access the hardware random number generator. Not available on all systems (see os.urandom() for details). “””

```
get_bits = None
get_bytes = None

def random(self):
    """Get the next random number in the range [0.0, 1.0).
       52 is the number of bits
    """
    return (1.0 * self.get_bits(52)) / (2 ** 52)

def getrandbytes(self, k):
    if k <= 0:
        raise ValueError('number of bytes must be greater than zero')
    if k != int(k):
        raise TypeError('number of bytes should be an integer')
    return self.get_bytes(k)

def getrandbits(self, k):
    """getrandbits(k) -> x.  Generates a long int with k random bits."""
    if k <= 0:
        raise ValueError('number of bits must be greater than zero')
    if k != int(k):
        raise TypeError('number of bits should be an integer')
    return self.get_bits(k)

def _stub(self, *args, **kwds):
    "Stub method.  Not used for a system random number generator."
    return None
seed = jumpahead = _stub

def _notimplemented(self, *args, **kwds):
    "Method should not be called for a system random number generator."
    raise NotImplementedError('System entropy source does not have state.')
getstate = setstate = _notimplemented
```

if HAS_RAND == 1: class RdRandom(BaseRandom): get_bits = rdrand_get_bits get_bytes = rdrand_get_bytes

if HAS_SEED == 1: class RdSeedom(BaseRandom): get_bits = rdseed_get_bits get_bytes = rdseed_get_bytes

Does it change random numbers from C compilation to floats? If so, can you modify that part to return integers?

Bing

Yes, the `random` method in the `BaseRandom` class is converting the random bits obtained from the `get_bits` method into a float in the range [0.0, 1.0). This is done by the line:

```
return (1.0 * self.get_bits(52)) / (2 ** 52)
```

If you want this method to return an integer instead, you can simply return the result of `get_bits` directly. However, you need to be aware of the range of the returned integer. The `get_bits` method is called with the argument `52`, which means it will return an integer with up to 52 bits. If you want a smaller range, you can use the modulo operation as I described in my previous message.

Here's how you can modify the `random` method to return integers:

```
def random(self):
    """Get the next random number as an integer."""
    return self.get_bits(52)
```

And if you want to limit the range to, say, 0 to 99, you can do:

```
def random(self):
    """Get the next random number as an integer in the range [0, 100)."""
    return self.get_bits(52) % 100
```

Please note that this will change the behavior of the `random` method in all classes that inherit from `BaseRandom`, including `RdRandom` and `RdSeedom`. If you only want to change the behavior in one of these classes, you should override the `random` method in that class instead. Also, remember to update the docstring to reflect the new behavior. Let me know if you need further assistance!

User

What of I want maximum number of bits instead of 52?

Bing

If you want to use the maximum number of bits, you can call `get_bits` with the argument `64` because `get_bits` returns a 64-bit integer (`uint64_t`). Here's how you can modify the `random` method:

```
def random(self):
    """Get the next random number as an integer."""
    return self.get_bits(64)
```

This will return a random 64-bit integer. If you want a smaller range, you can use the modulo operation. For example, to get an integer in the range [0, 100), you can do:

```
def random(self):
    """Get the next random number as an integer in the range [0, 100)."""
    return self.get_bits(64) % 100
```

Please note that the range of a 64-bit integer is from 0 to 18,446,744,073,709,551,615 (2^64 - 1). If you don’t limit the range, the numbers can be very large. Let me know if you need further assistance!

User

My xlwings py code is as under: rng = rdrand.RdRandom() @xw.func def generate_random_numbers_rdrand(num): # Generate the random numbers random_numbers = [[int(rng.random())] for _ in range(int(num))] return random_numbers @xw.sub def main(): # Expose the UDF to Excel utils.excel.register_udfs([generate_random_numbers])

Run the main function when the script is executed

if **name** == “**main**”: xw.Book().set_mock_caller() main()

and my output with above changes in C source file of rdrand is as under:

15287020628689600000 5686109473770420000 14233453878084000000 9659707329408970000 2569174844923100000 18330350923399900000 17495534708975500000 6216864844887430000 10124985257837700000 14149144490329700000 10231679258697400000

You can see trailing zeros. Can you modify the above code of xlwings to generate 15 digits length random numbers uniformly?

Bing

Sure, I can help with that. To generate 15-digit random numbers uniformly, you can use the `random.randint` function from the Python standard library, which generates a random integer uniformly in the given range. Here’s how you can modify your `generate_random_numbers_rdrand` function:

```
import random

@xw.func
def generate_random_numbers_rdrand(num):
    # Generate the random numbers
    random_numbers = [[random.randint(10**14, 10**15 - 1)] for _ in range(int(num))]
    return random_numbers
```

This will generate random numbers with exactly 15 digits. The range `10**14` to `10**15 - 1` ensures that the numbers have at least 15 digits but less than 16 digits.

Please note that this code doesn’t use the `rdrand` package anymore, because Python’s `random` module is sufficient for this task. If you still want to use `rdrand`, you could do something similar, but you would need to handle the case where `rng.random()` returns a number with less than 15 digits. Let me know if you need further assistance!

User

error: “NameError: name ‘random’ is not defined. Did you forget to import ‘random’ ^^^^^^ random_numbers = [[random.randint(10**14, 10**15 - 1)] for _ in range(int(num))] File ““D:\Onedrive\OneDrive - 0yt2k\xlwings_trial.py””, line 151, in generate_random_numbers_rdrand ^^^^^^^^^^^ ret = func(*args) File ““D:\Programs\Python\Lib\site-packages\xlwings\udfs.py””, line 525, in call_udf ^^^ res = call_udf(script, fname, args, this_workbook, FromVariant(caller)) File ““D:\Programs\Python\Lib\site-packages\xlwings\com_server.py””, line 235, in CallUDF ^^^^^^^^^^^^ return func(*args) File ““D:\Programs\Python\Lib\site-packages\win32com\server\policy.py””, line 639, in invokeex ^^ return S_OK, -1, self.invokeex(dispid, lcid, wFlags, args, None, None) File ““D:\Programs\Python\Lib\site-packages\win32com\server\policy.py””, line 310, in invoke ^^^ return self.invoke(dispid, lcid, wFlags, args) File ““D:\Programs\Python\Lib\site-packages\win32com\server\policy.py””, line 305, in Invoke”

Bing

I apologize for the confusion. The error message indicates that the `random` module is not imported in your Python script. You can fix this by adding `import random` at the beginning of your script. Here's how your code should look:

```
import random
import xlwings as xw
from xlwings import utils
import rdrand

rng = rdrand.RdRandom()

@xw.func
def generate_random_numbers_rdrand(num):
    # Generate the random numbers
    random_numbers = [[random.randint(10**14, 10**15 - 1)] for _ in range(int(num))]
    return random_numbers

@xw.sub
def main():
    # Expose the UDF to Excel
    utils.excel.register_udfs([generate_random_numbers_rdrand])

# Run the main function when the script is executed
if __name__ == "__main__":
    xw.Book().set_mock_caller()
    main()
```

This should resolve the `NameError: name 'random' is not defined` error. Let me know if you need further assistance!

User

But why you are using random. I want to generate using rdrand?

Bing

I apologize for the confusion. If you want to use the `rdrand` package to generate random numbers, you can modify the `generate_random_numbers_rdrand` function as follows:

```
@xw.func
def generate_random_numbers_rdrand(num):
    # Generate the random numbers
    random_numbers = [[rng.random() % (10**15)] for _ in range(int(num))]
    return random_numbers
```

This will generate random numbers with up to 15 digits using the `rdrand` package. The `rng.random() % (10**15)` line generates a random number and then uses the modulo operation to limit it to 15 digits. Please note that this might not generate a uniform distribution of 15-digit numbers, because `rng.random()` returns a 64-bit number which is then reduced to 15 digits. If you need a uniform distribution of 15-digit numbers, you would need a more complex method. Let me know if you need further assistance!

User

Instead of func and sub with main can you not combine into one function (in xw)?

Bing

Yes, you can combine the function and subroutine into one function in `xlwings`. However, please note that this will change the behavior of your code. The `@xw.func` decorator is used to expose a Python function to Excel as a User Defined Function (UDF), while the `@xw.sub` decorator is used to expose a Python function to Excel as a macro.

If you want to combine them into one function, you could do something like this:

```
import xlwings as xw
from xlwings import utils
import rdrand

rng = rdrand.RdRandom()
```

```
@xw.func
def generate_random_numbers_rdrand(num):
    # Generate the random numbers
    random_numbers = [[rng.random() % (10**15)] for _ in range(int(num))]

    # Expose the UDF to Excel
    utils.excel.register_udfs([generate_random_numbers_rdrand])

    return random_numbers

# Run the main function when the script is executed
if __name__ == "__main__":
    xw.Book().set_mock_caller()
    generate_random_numbers_rdrand(10)
```

In this code, the `generate_random_numbers_rdrand` function generates the random numbers and also registers itself as a UDF. When the script is executed, it calls `generate_random_numbers_rdrand` with an argument of `10`, generating 10 random numbers.

Please note that this might not be the best practice, as it's generally better to separate concerns by having different functions for different tasks. Also, this will register the UDF every time `generate_random_numbers_rdrand` is called, which might not be what you want. Let me know if you need further assistance!

User

Can you not change:

generate_random_numbers_rdrand(10)

To take argument from generate_random_numbers_rdrand(num):

Bing

Yes, you can modify the code to take the argument from `generate_random_numbers_rdrand(num)`. However, when the script is executed (`if __name__ == "__main__":`), you need to provide a specific number as an argument to the function. Here's how you can modify the code:

```
import xlwings as xw
from xlwings import utils
import rdrand

rng = rdrand.RdRandom()

@xw.func
def generate_random_numbers_rdrand(num):
    # Generate the random numbers
    random_numbers = [[rng.random() % (10**15)] for _ in range(int(num))]

    # Expose the UDF to Excel
    utils.excel.register_udfs([generate_random_numbers_rdrand])

    return random_numbers

# Run the main function when the script is executed
if __name__ == "__main__":
    xw.Book().set_mock_caller()
    num = 10  # You can change this number to generate a different number of random numbers
    generate_random_numbers_rdrand(num)
```

In this code, the `num` variable is set to `10`, and then `generate_random_numbers_rdrand(num)` is called with `num` as an argument. This will generate 10 random numbers when the script is executed. You can change the value of `num` to generate a different number of random numbers. Let me know if you need further assistance!

User

But num will be supplied from excel not to be hardcoded in function.

Bing

I see, you want to supply the number of random numbers to generate (`num`) from Excel. In that case, you don't need to modify the `if __name__ == "__main__":` section. When you call the `generate_random_numbers_rdrand` function from Excel as a UDF, Excel will supply the `num` argument.

Here's how your code should look:

```
import xlwings as xw
from xlwings import utils
import rdrand

rng = rdrand.RdRandom()

@xw.func
def generate_random_numbers_rdrand(num):
    # Generate the random numbers
    random_numbers = [[rng.random() % (10**15)] for _ in range(int(num))]

    # Expose the UDF to Excel
    utils.excel.register_udfs([generate_random_numbers_rdrand])

    return random_numbers

# Run the main function when the script is executed
if __name__ == "__main__":
    xw.Book().set_mock_caller()
```

In Excel, you can then use the `generate_random_numbers_rdrand` function as a UDF. For example, if you want to generate 10 random numbers, you can enter `=generate_random_numbers_rdrand(10)` in a cell. Let me know if you need further assistance!

User

bing ai

Bing

If you're looking for answers, our new search engine is just what you need! We've taken Microsoft Bing's rich results to the next level with powerful artificial intelligence.

User

error after above change: "AttributeError: module 'xlwings.utils' has no attribute 'excel' ^^^^^^^^^^^^ utils.excel.register_udfs([generate_random_numbers_rdrand]) File ""“D:\Onedrive\OneDrive - 0yt2k\xlwings_trial.py""", line 154, in generate_random_numbers_rdrand ^^^^^^^^^^^^ ret = func(*args) File ""“D:\Programs\Python\Lib\site-packages\xlwings\udfs.py""", line 525, in call_udf ^^^ res = call_udf(script, fname, args, this_workbook, FromVariant(caller)) File ""“D:\Programs\Python\Lib\site-packages\xlwings\com_server.py""", line 235, in CallUDF ^^^^^^^^^^^^ return func(*args) File ""“D:\Programs\Python\Lib\site-packages\win32com\server\policy.py""", line 639, in invokeex ^^ return S_OK, -1, self.invokeex(dispid, lcid, wFlags, args, None, None) File ""“D:\Programs\Python\Lib\site-packages\win32com\server\policy.py""", line 310, in invoke ^^ return self.invoke(dispid, lcid, wFlags, args) File ""“D:\Programs\Python\Lib\site-packages\win32com\server\policy.py""", line 305, in Invoke"

Bing

I apologize for the confusion. The error message indicates that the `xlwings.utils` module does not have an attribute named `excel`. This is because the `register_udfs` function is actually part of the `xlwings.udfs` module, not the `xlwings.utils` module.

Here's how you can modify your code:

```
import xlwings as xw
from xlwings import udfs
import rdrand
```

```python
rng = rdrand.RdRandom()

@xw.func
def generate_random_numbers_rdrand(num):
    # Generate the random numbers
    random_numbers = [[rng.random() % (10**15)] for _ in range(int(num))]

    return random_numbers

# Register the UDF to Excel
udfs.register_udfs([generate_random_numbers_rdrand])

# Run the main function when the script is executed
if __name__ == "__main__":
    xw.Book().set_mock_caller()
```

In this code, I've imported the `xlwings.udfs` module and used `udfs.register_udfs` to register the UDF to Excel. This should resolve the `AttributeError: module 'xlwings.utils' has no attribute 'excel'` error. Let me know if you need further assistance!

User

"AttributeError: module 'xlwings.udfs' has no attribute 'register_udfs' ^^^^^^^^^^^^^^^^^^ udfs.register_udfs([generate_random_numbers_rdrand]) File ""\"D:\Onedrive\OneDrive - 0yt2k\xlwings_trial.py""", line 155, in generate_random_numbers_rdrand ^^^^^^^^^^^ ret = func(*args) File ""\"D:\Programs\Python\Lib\site-packages\xlwings\udfs.py""", line 525, in call_udf ^^ res = call_udf(script, fname, args, this_workbook, FromVariant(caller)) File ""\"D:\Programs\Python\Lib\site-packages\xlwings\com_server.py""", line 235, in CallUDF ^^^^^^^^^^^ return func(*args) File ""\"D:\Programs\Python\Lib\site-packages\win32com\server\policy.py""", line 639, in invokeex ^^ return S_OK, -1, self.invokeex(dispid, lcid, wFlags, args, None, None) File ""\"D:\Programs\Python\Lib\site-packages\win32com\server\policy.py""", line 310, in invoke ^^ return self.invoke(dispid, lcid, wFlags, args) File ""\"D:\Programs\Python\Lib\site-packages\win32com\server\policy.py""", line 305, in Invoke"

Bing

I apologize for the confusion. The `xlwings.udfs` module does not have a `register_udfs` function. The `@xw.func` decorator automatically makes the function available in Excel as a User Defined Function (UDF) when the Python script is imported in Excel using the `xlwings` Excel add-in.

Here's how your code should look:

```python
import xlwings as xw
import rdrand

rng = rdrand.RdRandom()

@xw.func
def generate_random_numbers_rdrand(num):
    # Generate the random numbers
    random_numbers = [[rng.random() % (10**15)] for _ in range(int(num))]
    return random_numbers

# Run the main function when the script is executed
if __name__ == "__main__":
    xw.Book().set_mock_caller()
```

In Excel, you can then use the `generate_random_numbers_rdrand` function as a UDF. For example, if you want to generate 10 random numbers, you can enter `=generate_random_numbers_rdrand(10)` in a cell. Let me know if you need further assistance!

User

Can you change to uniformly generate numbers of 15 digit length?

Bing

To generate uniformly distributed 15-digit random numbers, you can modify the `generate_random_numbers_rdrand` function as follows:

```
@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_numbers
```

This will generate random numbers with exactly 15 digits using the `rdrand` package. The `rng.random() % (10**15 - 10**14) + 10**14` line generates a random number and then uses the modulo operation to limit it to 15 digits, and adds `10**14` to ensure that it's not less than 15 digits.

Please note that this might not generate a uniform distribution of 15-digit numbers, because `rng.random()` returns a 64-bit number which is then reduced to 15 digits. If you need a uniform distribution of 15-digit numbers, you would need a more complex method. Let me know if you need further assistance!

User

You said: Please note that this might not generate a uniform distribution of 15-digit numbers, because rng.random() returns a 64-bit number which is then reduced to 15 digits. If you need a uniform distribution of 15-digit numbers, you would need a more complex method. Let me know if you need further assistance!

I want to know what is that complex method?

Bing

To generate uniformly distributed 15-digit random numbers, you can use a method that generates each digit independently. This ensures that all 15-digit numbers are equally likely. Here's how you can modify the `generate_random_numbers_rdrand` function:

```
@xw.func
def generate_random_numbers_rdrand(num):
    # Generate the random numbers
    random_numbers = [[int(''.join(str(rng.random() % 10) for _ in range(15)))] for _ in range(int(num))]
    return random_numbers
```

In this code, the inner loop `str(rng.random() % 10) for _ in range(15)` generates each digit independently, and `''.join(...)` combines them into a 15-digit number. The `int(...)` function then converts the number from a string to an integer.

Please note that this method assumes that `rng.random()` generates uniformly distributed random numbers between 0 and 9. If this is not the case, you would need to adjust the code accordingly. Let me know if you need further assistance!

User

Still after above I find numbers of length 12. Can you check the above code?

Bing

I apologize for the confusion. The issue is that `rng.random()` generates a 64-bit random number, not a random digit. To generate a random digit, you need to take `rng.random()` modulo 10. However, because `rng.random()` returns a large number, taking it modulo 10 does not give a uniform distribution of digits.

A better approach would be to generate a random number between 0 and 999,999,999,999,999 (15 digits), then add 1,000,000,000,000,000 (also 15 digits) to it. This ensures that the result is a uniformly distributed random number with exactly 15 digits.

Here's how you can modify the `generate_random_numbers_rdrand` function:

```
@xw.func
def generate_random_numbers_rdrand(num):
    # Generate the random numbers
    random_numbers = [[rng.random() % (10**15) + 10**15] for _ in range(int(num))]
    return random_numbers
```

This code generates a random number between 0 and 999,999,999,999,999 using `rng.random() % (10**15)`, then adds 1,000,000,000,000,000 to it to ensure that it has exactly 15 digits. Let me know if you need further assistance!

User

Some problem. My output after above: 1222279113604630 1807642670749340 1990474624142190 1705261901586870 1962239782820180 1355319707424910 1257971138335160 All numbers starting from 1 as you can see.

Bing

I apologize for the confusion. The issue is that the `rng.random()` function generates a 64-bit random number, which is much larger than the 15-digit number we want. When we take this number modulo `10**15`, the result is not uniformly distributed.

To generate a uniformly distributed 15-digit number, we need to generate each digit independently. However, the `rdrand` package does not provide a way to generate a random digit directly. We would need to generate a random 64-bit number, convert it to a string, take the first digit, and repeat this process 15 times. This is quite inefficient and may not be feasible in your application.

Unfortunately, without a way to generate a random digit or a random number in a specific range directly, it's not possible to generate a uniformly distributed 15-digit number using the `rdrand` package. You may want to consider using a different method to generate random numbers if you need them to be uniformly distributed and exactly 15 digits long. Let me know if you need further assistance!

User

With change as below:

random_numbers = [[int(rng.random() % (10**15 - 10**14)) + 10**14] for _ in range(int(num))]

I could generate random numbers in all ranges not necesarily starting with 1: Output sample:

738542421376963 511546626977245 909840913740729 772649948715066 935043594989449 502215216968825 530219139422948 861798590102775 513231800892028 498507029850796 408772402571902 735507791045399

Can you explain why. This command has been suggested by you only. Look at your previous responses.

Bing

The line of code `[[int(rng.random() % (10**15 - 10**14)) + 10**14] for _ in range(int(num))]` works as follows:

1. `rng.random()` generates a random 64-bit number.
2. `rng.random() % (10**15 - 10**14)` takes the remainder of the division of the random number by `10**15 - 10**14` (which is `900,000,000,000,000`). This gives a number in the range `[0, 900,000,000,000,000)`.
3. `int(rng.random() % (10**15 - 10**14)) + 10**14` adds `10**14` (which is `100,000,000,000,000`) to the result. This shifts the range to `[100,000,000,000,000, 1,000,000,000,000,000)`, which is the range of 15-digit numbers.

So this line of code generates a random 15-digit number. The reason you're seeing numbers that don't start with 1 is because the addition of `10**14` shifts the entire range up, so the first digit can be any number from 1 to 9. Let me know if you need further assistance!