SHONNER PRESS

diceroll Operations Manual

Release 3.0.1b

Shawn Driscoll

February 07, 2020

Contents

SHONNER PRESS

diceroll 3.0 is easy-to-use open source die rolling software. Written in Classic Python 2.5 and using a variety of IDEs, **diceroll 3.0** supports many gaming and RPG die rolling conventions.

diceroll 3.0 also supports logging, error reporting, and debugging of rolls made.

The free-to-use source is available at its GitHub repository.

This documentation explains how to install and use the **diceroll module** for your gaming projects.

Download the PDF or the EPUB

CHAPTER 1

Introduction

1.1 Preface

Back during the release of **diceroll 2.2**, I wanted to learn something new in regards to Python. Even though I use 2.5.4, there is still a lot about it that I have never delved into. Sphinx was something I had not really paid any mind to in the past. It was yet another one of those *need to know only* things about Python. Some things I'd get around to learning only when I had to, but only if it was part of something else that I had taken an interest in doing.

So somewhere in my discovering of PyMongo, I had been pointed to Sphinx and Jinja. They were both something about document generation. And since I had just learned about Pandas and CSV, I was in a data retrieval mood still.

In a nutshell, Sphinx is an EXE (generated during its install from an egg of .py files, which is still magic to me, and which took a great deal of time for me tracking down all the proper versions of requirements for it to even compile/run in Classic Python 2.5.4) that generates documents. Nothing too fancy. Just simple documents that could be read easily/quickly through any device using any viewer. And when I learned that Sphinx could read Python modules and produce documents from their `.__doc__` strings, I knew I just had to spend a couple days learning how all that stuff happens.

So basically, my Python dice rolling module has its own operations manual now. And some rabbit holes are worth their going into.

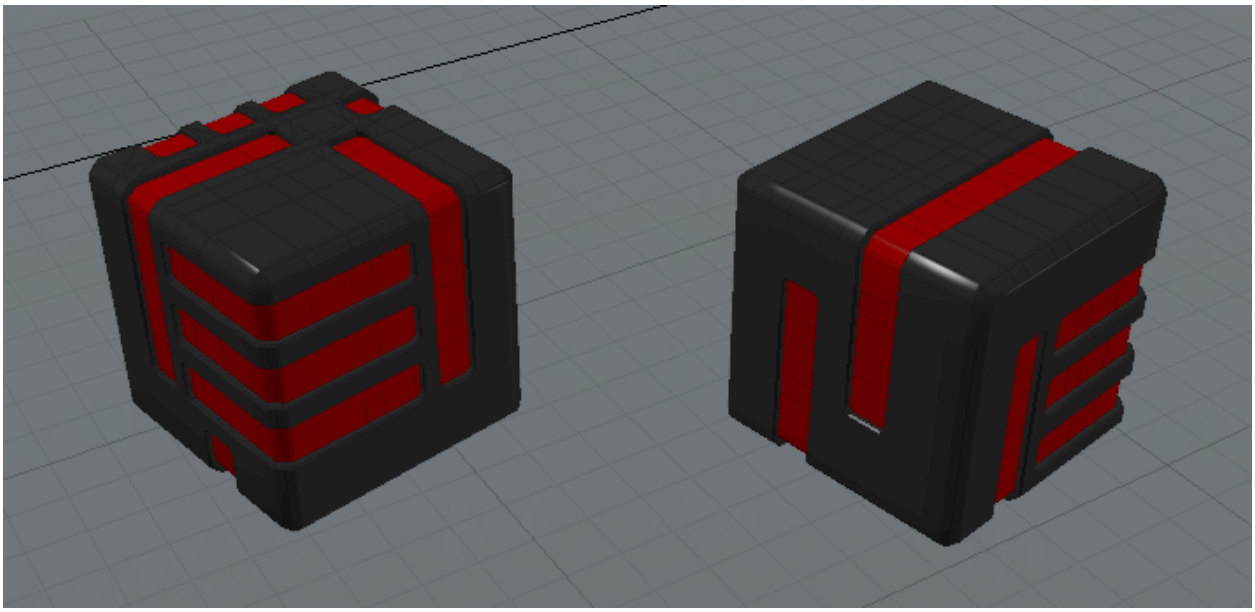
-Shawn

1.2 Requirements

- **Microsoft Windows**

  diceroll has been tested on Windows versions: XP, 7, 8, and 10. It has not been tested on MacOS or Linux.

- **Classic Python 2.5**



diceroll was written using the C implementation of Classic Python version 2.5.4. Also known as CPython. With some doing, this module could of course be re-written for Jython, PyPy, or IronPython.

Eclipse/PyDev, PyCharm, NetBeans, and IDLE all work fine for running this module.

- **colorama 0.2.7**

  Because CMD may have some colored text messages for debugging. The colorama code can be removed if it is not needed, however.

- **Your Game**

  diceroll is not a standalone program. It requires your game to make calls to it.

  Otherwise, no dice.

  (Update: As of version 2.4, **diceroll** can be used at the CMD prompt.)

Warning: **diceroll 3.0** will not work with **Python 2.6+**.

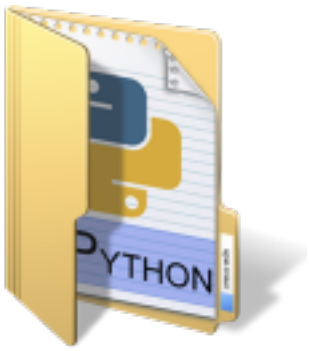
1.3 Installing Locally to Your Folder

Installing **diceroll 3.0** is as easy as always. Just copy `diceroll.py` into the same folder your code happens to be in.

Then add this line at (or near) the top of your code:

```
from diceroll import roll
```

1.4 Installing as a Package



If your code setup is different, in that you like to keep your function modules in a folder separate from your main code, you could copy `diceroll.py` into that folder.

Say you have a folder called `game_utils`, and assuming you have an `__init__.py` inside it, just copy `diceroll.py` into your `game_utils` folder and add this line near the top of your code:

```
from game_utils.diceroll import roll
```

1.5 Installing Automatically

New in version 2.3

Extract `diceroll_3.0.1b.zip` and start a CMD window at the folder location of the `setup.py` file. At the CMD prompt you can type:

```
setup.py install
```

or:

```
python setup.py install
```

depending on if your computer knows how to open .py files or not.

Note: During the installation process, a `Python25\Lib\site-packages\game_utils` folder will be created. It will contain `__init__.py` and `diceroll.py` if your Python doesn't have `setuptools` installed. Otherwise, an .egg file called `diceroll-3.0.1b-py2.5.egg` will be created and copied into the `Python25\Lib\site-packages` folder.

No matter the automated installation that your Python performed, importing will be the same:

```
from game_utils.diceroll import roll
```

Some ways to see if the `diceroll` module was installed correctly is by typing:

```
>>> print roll('info')
('3.0', 'roll(), release version 3.0.1b for Classic Python 2.5.4')
>>> print roll.__doc__
    The dice types to roll are:
        '4dF', 'D2', 'D3', 'D4', 'D6', 'D8', 'D09', 'D10',
        'D12', 'D20', 'D30', 'D099', 'D100', 'D66', 'DD',
        'FLUX', 'GOODFLUX', 'BADFLUX', 'BOON', 'BANE'
    Some examples are:
    roll('D6') or roll('1D6') -- roll one 6-sided die
    roll('2D6') -- roll two 6-sided dice
    roll('D09') -- roll a 10-sided die (0 - 9)
    roll('D10') -- roll a 10-sided die (1 - 10)
    roll('D099') -- roll a 100-sided die (0 - 99)
    roll('D100') -- roll a 100-sided die (1 - 100)
    roll('D66') -- roll for a D66 chart
    roll('FLUX') -- a FLUX roll (-5 to 5)
    roll('3D6+6') -- add +6 DM to roll
    roll('4D4-4') -- add -4 DM to roll
    roll('2DD+3') -- roll (2D6+3) x 10
    roll('BOON') -- roll 3D6 and keep the higher two dice
    roll('4dF') -- make a FATE roll
    roll('info') -- release version of program
    An invalid roll will return a 0.
```

CHAPTER 2

diceroll Tutorial

2.1 Rolling the Dice

Once `diceroll.py` is installed and your code is able to import the module, its `roll()` function can be used right away. This function returns an integer, by the way. So it can be used as any other integer would be used. But first, we must give this function a value to work from.

`roll`(*dice*)

> *dice* = a string of three ordered concatenated values:
>
> *number_of_dice* + *dice_type* + *dice_roll_modifier*
>
> As examples:
> *dice* = '2' + 'D10' + '-2'
> *dice* = str(3) + 'D6' + '+2'
> *dice* = 'FLUX'
>
> *dice_roll_modifier* must include a '+' or '-' with its value.

Note that both *number_of_dice* and *dice_roll_modifier* are optional, and may not even be used by some *dice_type* rolls.

Those of you that have used dice rolling programs before will notice that something is different. And that is, `roll()` uses a string for its input:

```
>>> die1 = roll('1D6')
>>> die2 = roll('1d6')
>>> dice = '3D4+1'
>>> print die1, die2+4, roll(dice)
3, 6, 9
```

The return values from `roll()` are always integer.

New in version 2.2

Notice that the inputted string values can be upper or lower case.

The dice types to roll are:

D3, D4, D6, D8, D10, D12, D20, D30, D100, D66, DD, FLUX, GOODFLUX, and BADFLUX

New in version 2.3

Three additional dice types are now available:

BOON, BANE, and D2

Note: You may recognize some of these dice types from various tabletop role-playing games. Not all dice types are covered by **diceroll**. However, more are planned for in future releases.

---

diceroll uses a simple standard when it comes to rolling various dice types.

Some examples are:

```
roll('D6') or roll('1D6') # roll one 6-sided die
roll('2D6') # roll two 6-sided dice
roll('D09') # roll a 10-sided die (0 - 9)
roll('D10') # roll a 10-sided die (1 - 10)
roll('D099') # roll a 100-sided die (0 - 99)
roll('D100') # roll a 100-sided die (1 - 100)
roll('D66') # roll for a D66 chart
roll('FLUX') # a FLUX roll (-5 to 5)
roll('3D6+6') # add +6 DM to roll
roll('4D4-4') # add -4 DM to roll
roll('2DD+3') # roll (2D6+3) x 10
roll('BOON') # roll 3D6 and keep the higher two dice
roll('4dF') # make a FATE roll
```

Deprecated in version 1.9.

D00 has been replaced with D100.

New in version 2.4

diceroll can now be used directly at a CMD prompt:

```
C:\>diceroll.py roll('2d6-2')

Your 2D6-2 roll is 10.
```

(continues on next page)

(continued from previous page)

```
C:\>diceroll.py 2d6-2

Your 2D6-2 roll is 7.
```

Note: Typing `diceroll.py -h` will provide some help.

New in release 2.4.1

A TEST roll that calculates percentages for 2D6 has been added:

```
>>> roll('test')
      6x6 Roll Chart Test
     1    2    3    4    5    6
1  262  296  250  292  292  241
2  270  315  299  236  279  261
3  295  274  288  274  291  295
4  273  284  279  276  249  273
5  293  280  291  276  280  283
6  270  276  282  272  273  280
           6x6 Roll Chart Percentage
        1       2       3       4       5       6
1   2.62%   5.66%   8.60%  11.38%  13.93%  16.23%
2   5.66%   8.60%  11.38%  13.93%  16.23%  13.95%
3   8.60%  11.38%  13.93%  16.23%  13.95%  11.02%
4  11.38%  13.93%  16.23%  13.95%  11.02%   8.25%
5  13.93%  16.23%  13.95%  11.02%   8.25%   5.56%
6  16.23%  13.95%  11.02%   8.25%   5.56%   2.80%
```

The roll will return a list of percentages for 2-12 rolled.

New in release 2.4.2

D09 rolls will generate a range of 0 - 9.

New in release 2.4.3

D99 rolls will generate a range of 0 - 99.

Fixed in release 2.4.7

Minor fixes with input spacing, and logging any negative dice rolled.

New in version 3.0

D2 rolls now generate a range of 0 - 1. The 4dF roll type for FATE has been added.

CHAPTER 3

Using roll() in Your Own Code

3.1 For Simple Die Rolls

Sample Outputting of Die Rolls:

```
# import the roll() module
from diceroll import roll

# enter the roll type to be made
number_of_dice = raw_input('Number of dice to roll? ')
dice_type = raw_input('Dice type? ')
dice_roll_modifier = raw_input('DM? ')

# make sure that there is a plus or minus sign in the DM string
if dice_roll_modifier[0] <> '-' and dice_roll_modifier[0] <> '+':
    dice_roll_modifier = '+' + dice_roll_modifier

# concatenate the values for the dice string
dice = number_of_dice + dice_type + dice_roll_modifier
```

(continues on next page)

(continued from previous page)

```
print
print 'Rolling', dice

# do 20 rolls
for i in range(20):
    print 'You rolled a %d' % roll(dice)
```

3.2 For Probabilites

Sample Task Resolution:

```
# import the roll() module
from diceroll import roll

# Enter your character's chances to succeed at a task
skilled = raw_input('Is your character trained for the task ([y]/n)? ')
if skilled == 'n':
    die_mod = -3
else:
    print "Enter your character's skill level"
    die_mod = int(raw_input('(0 to 4)? '))
print 'Enter the difficulty of the task'
difficulty = int(raw_input('(Impossible: -6 to Easy: +6)? '))

# The player must roll an 8 or higher for their character to succeed
dice_roll = roll('2D6') + die_mod + difficulty
print
print 'You rolled:', dice_roll
if dice_roll >= 8:
    print 'Your character succeeds with the task.'
    if dice_roll - 8 >= 6:
        print 'Your character saved everyone.'
else:
    print 'Your character fails at the task.'
    if dice_roll - 8 < -3:
        print 'Your character becomes injured.'
    if dice_roll - 8 < -6:
        print 'Your character died from injuries!'
```

3.3 For Repairing Game Code

Often times, game code will be downloaded or found that contains incorrect `randint()` calls for rolling two 6-sided dice. A line such as:

```
world_size = randint(2, 12) - 2
```

Easily becomes:

```
world_size = roll('2d6') - 2
```

3.4 Encountering Errors

Entering an invalid string for `roll()` will return an error message, as well as a value of 0 from the function:

```
print roll('3d')
```

Error: ** DICE ERROR! '3D' is unknown **

0

CHAPTER 4

Debugging diceroll

diceroll 3.0 keeps a log file of any dice rolls made during its last run. You will find `diceroll.log` in the `Logs` folder it creates if one isn't there already. In the file you will see mentions of dice being rolled. **diceroll** uses a default logging mode of `INFO` which isn't that verbose.

```
diceroll_log.setLevel(logging.INFO)
```

Your INFO logging will output as:

. . . INFO diceroll - Logging started.
. . . INFO diceroll - roll() v3.0 started, and running. . .
. . . INFO diceroll - 3D4 = 3D4+0 = 10

Changing **diceroll's** logging mode to `DEBUG` will record debugging messages in the `Logs\diceroll.log` file.

```
diceroll_log.setLevel(logging.DEBUG)
```

Your DEBUG logging will output as:

. . . INFO diceroll - Logging started.
. . . INFO diceroll - roll() v3.0 started, and running. . .
. . . DEBUG diceroll - Asked to roll 3D4:

. . . DEBUG diceroll - Using three 4-sided dice. . .
. . . DEBUG diceroll - Rolled a 4
. . . DEBUG diceroll - Rolled a 2
. . . DEBUG diceroll - Rolled a 2
. . . INFO diceroll - 3D4 = 3D4+0 = 8

Note: Running **diceroll** in `DEBUG` mode may create a log file that will be too huge to open. A program of yours left running for a long period of time could create millions of lines of recorded log entries. Fortunately, `diceroll.log` is reset each time your program is run. Also, any errors encountered will be recorded as `ERROR` in the log file, no matter which logging mode you've chosen to use.

If your own code has logging enabled for it, be sure to let **diceroll** know by changing `your_code_name_here` to the name of the program you're calling `roll()` from.

```
log = logging.getLogger('your_code_name_here.diceroll')
```

CHAPTER 5

Designer's Notes

5.1 In the Beginning

One of the first things I do when learning a new language is to discovery how it generates random numbers. Older computer languages from the '70s had their own built-in random number generators. Technically, they were pseudo-random number generators. But technically, I wanted to program my Star Trek games anyway no matter what they were called.

In the '80s, I would discover that not all computer languages came with random number generators built in. Many didn't have such a thing unless some external software library was installed. Both FORTRAN and C couldn't do random anything out of the box. A math library had to be picked from the many that were out there. And if none were available, a computer class on campus was available to teach you how to program your own random number generator from scratch.

By the '90s, random number generators were pretty much standardized as for as how accurately random they were. And they were included in standard libraries for various languages. By the time Python was being developed, the C language used to write Python had very robust random number generators. And because Python was written in C, it just made sense for it to make use of C libraries.

For those that are curious, **diceroll** uses the `random.randint()` module that comes with CPython. There are stronger random generators out there now, with NumPy being one of them. But at the time of designing **diceroll**, I didn't quite understand how-all NumPy worked, or what version of it to install. And for rolling dice, the built-in random number generator would be just fine.

5.2 Lessons Learned

In the past, when I needed a random number from 1 to say 6 (see 6-sided dice), I would use `INT(RND(1)*6) + 1`. And I would be used to doing it that way for probably 15 years or so, because that is how most BASIC languages did things. Other languages like C required me to whip out the 80286 System Developer's 3-ring binder to find out how `srand()` and `rand()` worked, and under what circumstances.

Fast-forward 20 years, and I'm learning CPython without knowing the difference between a CPython or an RPython or any other Python out there. I figured Python was the same all over, even though I had a feeling Linux did things

differently because of its filepath naming and OS commands. And of course, the first thing I had to try was Python's `random` module, as well as its ugly-looking `randint()`.

Right away I noticed the way Python "loaded" modules was going to be a learning experience. I hadn't really programmed anything huge since my TANDY Color Computer 3 days running OS-9 Level II and programming in BASIC09 (https://en.wikipedia.org/wiki/BASIC09). Python would reveal different ways of importing modules the more I read about them, and the more code I poured over.

I would soon find that:

```
import random

print random.randint(1, 6) # roll a 6-sided die
```

Was the same thing as:

```
from random import randint

print randint(1, 6) # roll a 6-sided die
```

Which looked a bit cleaner. But I was debating if I wanted to use `randint()` at all in my normal coding.

So while I was learning how to write my own functions, as well as how to go about importing them, I came up with an idea for **diceroll**. It would included a `roll()` function, and a `die_rolls()` function as a "side effect." Even though `die_rolls()` had no error-checking, `roll()` would call it after doing its own error-checking.

I was trying to avoid using:

```
from diceroll import die_rolls

print die_rolls(6, 2) # roll two 6-sided dice
```

For my dice rolls, I wanted something more readable. Something like:

```
from diceroll import roll

print roll('2D6') # roll two 6-sided dice
```

It was almost less typing, which I thought was great because I was going to be typing this function a lot for a Python project I had in mind. And it would be a lot easier to spot what kind of rolls were being made in my code. And the simple addition or subtraction of DMs to such a roll was making the function more appealing:

```
print roll('2D6+3') # roll two 6-sided dice and add a DM of +3 to it
```

5.3 The Channel 1

diceroll was written years ago. The code is used by both my TravCalc and TravGen apps, and gets looked at by GitHub visitors who google-by now and again. But not many programmers will use the code because of the simple fact that Python is now version 3.6+ something. So **diceroll**, along with a slew of other pre-Python 2.6 era modules, are the Channel 1 stations in the room that no TV can possibly watch.

It really comes down to a philosophy. I waited on learning Python until a version was released where I could say, *"This is Python."* Or say, *"This is what Python should be."* Something like that.

And for me, it was Classic Python 2.5.4 when I said such things. Python 2.6 books were showing up in stores. And there were already differences being found between it and the Python that I was using. Python had become this huge thing. And non-programmers were being attracted to it for their own reasons. And that was all fine. Python 2.7, 3.0,

etc., were seeing lots of new talent joining their mix. They were taking Python to places it hadn't been to. And more and more people were doing Python because of it.

Python is trying to be all things to all programmers these days. And it has become less of Python in doing so. I am not a functional programmer. Never have been. But a lot of people are. And Python now serves them very well. I'm often told, *"Python now does things this way."* But it is ways that I don't see myself using.

People are altering **diceroll** so that it works in their Python, just as I am altering their uploaded code so that it works in my Python. If I wanted my code to reach more people, of course I would have to program using the latest greatest Python. But there is a certain individuality lost in doing that.

I believe the next great computer programming language will be the one that remains true to its nature/design as it grows. And doesn't split the party as it grows.

Shawn Driscoll
October 3rd, 2017
US, California

CHAPTER 6

diceroll Module

roll(*number_of_dice* + *dice_type* + *dice_roll_modifier*)

`roll()` accepts a string value made up of three concatenated values, then returns an integer.

String values comes from *number_of_dice* + *dice_type* + *dice_roll_modifier*

Some examples are:
‘2’ + ‘D10’ + ‘-2’
str(3) + ‘D6’ + ‘+2’
‘FLUX’

dice_roll_modifier must include a ‘+’ or ‘-‘ with its value.

Note that both *number_of_dice* and *dice_roll_modifier* are optional, and may not even be used by some *dice_type* rolls.

CHAPTER 7

Glossary

80286 A CPU used by home computers in the mid-1980s.

BASIC09 A structured BASIC programming language dialect developed by Microware and Motorola for the then-new Motorola 6809 CPU and released in 1980. It was the best computer programming language until Python was invented.

CMD Command Prompt (CMD) is a command line interpreter program available in Windows 10, 8, 7, Vista, and XP. Command Prompt is similar in appearance to MS-DOS.

concatenation String concatenation is the operation of joining character strings end-to-end. For example, the concatenation of "iron" and "man" is "ironman".

C A computer programming language used to write a better computer programming language called Python.

CPython CPython is the default, most widely used implementation of the Python programming language. It is written in C.

D100 A 100-sided die. A sphere, basically. Rolled with caution.

debug The process of finding and resolving of defects that prevent correct operation of computer software or a system.

dice Small throwable objects with multiple resting positions, used for generating random numbers. Dice are suitable as gambling devices for games like craps and are also used in tabletop games.

diceroll A Python module available from this GitHub repository.

egg Eggs are to Pythons as Jars are to Java. Python eggs are a way of bundling additional information with a Python project, that allows the project's dependencies to be checked and satisfied at runtime, as well as allowing projects to provide plugins for other projects. The most common format is the '.egg' zipfile format, because it's a convenient one for distributing projects. All of the formats support including package-specific data, project-wide metadata, C extensions, and Python code.

errors Bugs that need to be squashed.

FORTRAN A computer programming language used to play Star Trek games in the 1970s.

game An activity engaged in for diversion or amusement. For computer games, it means no sweating.

IDE An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development.

integer An integer is what is more commonly known as a whole number. It may be positive, negative, or the number zero, but it must be whole.

log A log is a file that records events that occur as software runs. Logging is the act of keeping a log. In the simplest case, messages are written to a single logfile.

module A module is a part of a program. Programs are composed of one or more independently developed modules that are not combined until the program is linked.

no dice Used to refuse a request or indicate no chance of success.

Python 3.8+ A newfangled version of Python that's different from what Classic Python 2.5 programmers are used to.

rabbit hole Used to refer to a bizarre, confusing, or nonsensical situation or environment, typically one from which it is difficult to extricate oneself.

random The lack of pattern or predictability in events. A random sequence of events, symbols or steps has no order and does not follow an intelligible pattern or combination. Individual random events are by definition unpredictable, but in many cases the frequency of different outcomes over a large number of events (or "trials") is predictable.

Sphinx The Python software used to publish this operations manual.

string A string is a contiguous sequence of symbols or values, such as a character string (a sequence of characters) or a binary digit string (a sequence of binary values).

your own code Your own code is a Python program that you have already written to make calls to the `roll()` function.

CHAPTER 8

Open Source

8.1 MIT License

LICENSE AGREEMENT

8.2 Contact

Questions? Please contact shawndriscoll@hotmail.com

CHAPTER 9

About the Author

Shawn Driscoll is an American artist. Computers are his main creation tool. His many hobbies are in sync with his being a student of all sciences. Some of which are discussed in length on his YouTube channel.

CHAPTER 10

Indices and tables

- genindex
- search

Python Module Index

d

Index