SHONNER PRESS

PyDiceroll 3.2 Operations Manual

Release (2nd Edition)

Shawn Driscoll

05/21/2021

CONTENTS

1 Introduction **3**
- 1.1 Preface . . . 3
- 1.2 Requirements . . . 3
- 1.3 Installing Locally to Your Folder . . . 4
- 1.4 Installing as a Package . . . 5

2 What's New with PyDiceroll 3.2? **7**
- 2.1 Parsing . . . 7
- 2.2 Refactored for Python 3.9 . . . 7
- 2.3 The D5 Die . . . 7

3 PyDiceroll Tutorial **9**
- 3.1 Rolling the Dice . . . 9

4 Using roll() in Your Own Code **13**
- 4.1 For Simple Die Rolls . . . 13
- 4.2 For Probabilites . . . 14
- 4.3 For Repairing Game Code . . . 14
- 4.4 Encountering Errors . . . 15

5 Debugging PyDiceroll **17**

6 Alternate PyDiceroll Distributions **19**

7 Software Titles That Use PyDiceroll **21**

8 Designer's Notes **23**
- 8.1 In the Beginning . . . 23
- 8.2 Lessons Learned . . . 23
- 8.3 The Channel 1 . . . 24

9 PyDiceroll Module **27**

10 Glossary **29**

11 Open Source **31**
- 11.1 MIT License . . . 31
- 11.2 Contact . . . 31

12 FFE Agreement **33**

13 About the Author **35**

14 Indices and tables 37

Python Module Index 39

Index 41

SHONNER PRESS

PyDiceroll 3.2 is easy-to-use open source die rolling software. Written in Python 3.9.4 and using a variety of IDEs, **PyDiceroll 3.2** supports many gaming and RPG die rolling conventions.

PyDiceroll 3.2 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 **PyDiceroll module** for your gaming projects.

Download the PDF or the EPUB

CHAPTER

ONE

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 was using 2.5.4, there was 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 a pip command, which is still magic to me how *it just runs* in Python 3.9+) 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**

  PyDiceroll has been tested on Windows versions: 10. It has not been tested on MacOS or Linux.

- **Python 3.9+**

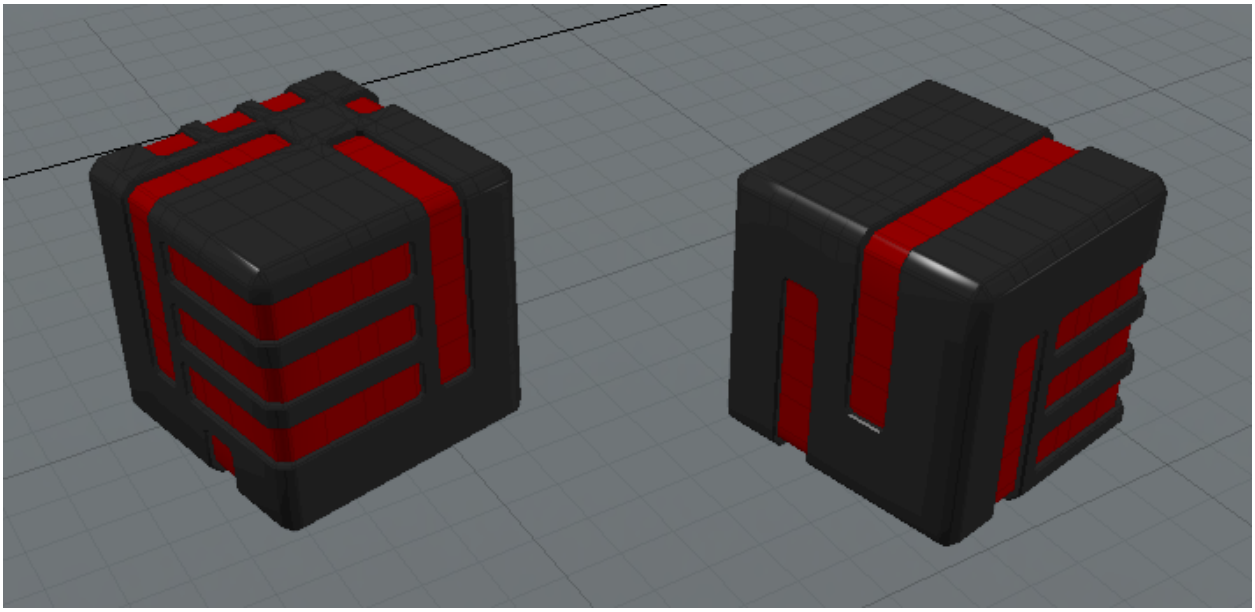
  PyDiceroll was written using the C implementation of Python version 3.9.4. Also known as CPython. With some doing, this module could of course be re-written for Jython, PyPy, or IronPython.

  Eclipse/PyDev, PyCharm, NetBeans, and IDLE all work fine for running this module. One of the easiest to use is Microsoft Visual Studio Code.

- **colorama 0.4.4**

  Because CMD may have some colored text messages for debugging. The colorama code can be removed if it is not needed, however. To install, just type `pip install colorama` at the CMD prompt.

- **Your Game**



PyDiceroll can be used as a standalone program. But where it shines though is when it's imported into a game of yours.

Warning: **PyDiceroll 3.2** will not work with **Python 2.7-**.

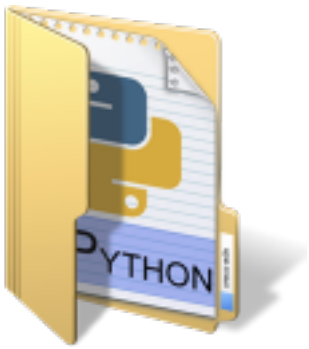
1.3 Installing Locally to Your Folder

Installing **PyDiceroll 3.2** is as easy as always. Just copy `PyDiceroll.py` into the same folder your code happens to be in.

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

```
from PyDiceroll 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 `PyDiceroll.py` into that folder.

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

```
from game_utils.PyDiceroll import roll
```

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

```
>>> print(roll('info'))
('3.2', 'roll(), release version 3.2.0b for Python 3.9.4')
>>> print(roll.__doc__)
    The dice types to roll are:
        '4dF', 'D2', 'D3', 'D4', 'D5', 'D6', 'D8', 'D09', 'D10',
        'D12', 'D20', 'D30', 'D099', 'D100', 'D66', 'DD',
        'FLUX', 'GOODFLUX', 'BADFLUX', 'BOON', 'BANE',
        and also Traveller5's 1D thru 10D rolls
    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('4D') -- make a Traveller5 4D roll
    roll('4dF') -- make a FATE roll
    roll('info') -- release version of program
    An invalid roll will return a 0.
```

CHAPTER

TWO

WHAT'S NEW WITH PYDICEROLL 3.2?

2.1 Parsing

The roll() function has improved parsing that allows for spaces from other program sources. Error-checking understands this and will even check for negative numbers of dice. This improved feature works whether **PyDiceroll** is being used in a Python program or at a CMD prompt.

2.2 Refactored for Python 3.9

PyDiceroll's code has been updated from 2.5 to 3.9 standards.

2.3 The D5 Die

The D5 has been added to **PyDiceroll**. It is basically a D10 divided by 2, much like how the D3 die is a D6 that is divided by 2.

CHAPTER

THREE

PYDICEROLL TUTORIAL

3.1 Rolling the Dice

Once `PyDiceroll.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')
```

(continues on next page)

(continued from previous page)

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

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

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

The dice types to roll are:

> **D2, D3, D4, D5, D6, D8, D09, D10, D12, D20, D30, D99, D100, D66, DD, BOON, BANE, FLUX, GOODFLUX, BADFLUX**, and **4dF**.

D09 rolls will generate a range of **0 - 9**.

D99 rolls will generate a range of **0 - 99**.

D2 rolls will generate a range of **0 - 1**.

The **4dF** roll type is for FATE mechanics.

Traveller5 uses **1D** thru **10D** rolls, depending on the difficulty of a task. DMs are supported.

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

PyDiceroll 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('4D') # make a Traveller5 4D roll
roll('4dF') # make a FATE roll
```

PyDiceroll can be used directly at a CMD prompt using:

The long form:

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

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

Or the short form:

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

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

Note: Typing `PyDiceroll.py -H` will provide some help.

A **TEST** roll that calculates percentages for 2D6 can be issued:

```
>>> 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.

CHAPTER

FOUR

USING ROLL() IN YOUR OWN CODE

4.1 For Simple Die Rolls

Sample Outputting of Die Rolls:

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

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

print()
print('Rolling', dice)

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

4.2 For Probabilites

Sample Task Resolution:

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

# Enter your character's chances to succeed at a task
skilled = 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(input('(0 to 4)? '))
print('Enter the difficulty of the task')
difficulty = int(input('(Simple: +2 to Impossible: +16)? '))

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

4.3 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
```

4.4 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('3d1'))
```

Error: ** DICE ERROR! '3D1' is unknown **

0

CHAPTER

FIVE

DEBUGGING PYDICEROLL

PyDiceroll 3.2 keeps a log file of any dice rolls made during its last run. You will find `PyDiceroll.log` in the `Logs` folder it creates if one isn't there already. In the file you will see mentions of dice being rolled. **PyDiceroll** uses a default logging mode of `INFO` which isn't that verbose.

```
PyDiceroll_log.setLevel(logging.INFO)
```

Your **INFO** logging will output as:

…INFO PyDiceroll - Logging started.
…INFO PyDiceroll - roll() v3.2 started, and running…
…INFO PyDiceroll - '3D4' = 3D4+0 = 10

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

```
PyDiceroll_log.setLevel(logging.DEBUG)
```

Your **DEBUG** logging will output as:

…INFO PyDiceroll - Logging started.
…INFO PyDiceroll - roll() v3.2 started, and running…
…DEBUG PyDiceroll - Asked to roll '3D4':
…DEBUG PyDiceroll - Using three 4-sided dice…
…DEBUG PyDiceroll - Rolled a 4
…DEBUG PyDiceroll - Rolled a 2
…DEBUG PyDiceroll - Rolled a 2
…INFO PyDiceroll - '3D4' = 3D4+0 = 8

Warning: Running **PyDiceroll** 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, `PyDiceroll.log` is reset each time your program is run.

Note: 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 **PyDiceroll** know by changing `your_logger_function_here` to the name of the logger function used by your program that is calling `roll()`. The original line in **PyDiceroll** looks like this:

```
log = logging.getLogger('your_logger_function_here.PyDiceroll')
```

So, if your own code has:

```
log = logging.getLogger('dungeoneer')
```

then in **PyDiceroll**, make

```
log = logging.getLogger('dungeoneer.PyDiceroll')
```

CHAPTER

SIX

ALTERNATE PYDICEROLL DISTRIBUTIONS

Often times, the **PyDiceroll** module is found in other formats. You may already have a copy of **PyDiceroll** that was distributed with another program you're using.

Besides its common `.py` format, **PyDiceroll** can be found in a `.pyd` format as well. This format is packaged as a dynamic link library, and will work the same way as the `.py` format. The format is typically bundled with the software that it was designed for.

Note: The `.pyd` format can only be imported and will not execute at a CMD prompt.

CHAPTER

SEVEN

SOFTWARE TITLES THAT USE PYDICEROLL

Here is a sample list of software titles, at the time of this writing, using **PyDiceroll**:

PyTravCalc 3.1.5

PyImperial CharGen 1.0.0

PyQt5-Dice-App

CHAPTER

EIGHT

DESIGNER'S NOTES

8.1 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, **PyDiceroll** 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 **PyDiceroll**, 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.

8.2 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 **PyDiceroll**. It would include 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 PyDiceroll 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 PyDiceroll 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
```

8.3 The Channel 1

diceroll was written years ago. The Classic Python 2.5 code was used by both my TravCalc and TravGen apps, and got looked at by GitHub visitors who would google-by now and again. But not many programmers will ever use the code because of the simple fact that Python is now version 3.9+ 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. Well… Python 2.5.4 was my Python.

I once said, *"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."* I hung onto Python 2.5.4 for as long as possible. For a good fifteen years. Or I should say for a *great* fifteen years. Because they were great. But most great things come with an ending to them.

And so it was, that yesterday I would uninstall Python 2.5.4 along with all its things. And today, I would begin the installation of Python 3.9.4. I guess one could say it was the terminated support for Python 2.x this year that nudged me, along with some of the older Python 3.x terminations as well. Even Python 3.9+ saw earlier Python 3's as dead weight (Python 3's that I didn't want to deal with either), such as 3.0, 3.1, 3.2, 3.3, 3.4, and 3.5. And now they are gone. And I can skip ahead to a refined version of Python 3 with no baggage from 2.6 or 2.7 whatsoever.

Shawn Driscoll
April 23rd, 2021
US, California

CHAPTER

NINE

PYDICEROLL 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

TEN

GLOSSARY

ASM Rarely is Assembly language programming done by hand. Rumors say it will make a huge comeback, once people stop using compilers.

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.

Bugs Coding errors (logic, run-time, or compile-time) that need fixing (debugging).

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.

COBOL A programming language that did not see the coming days of Python. For it was already a dead language.

compiler Converts the long-winded and verbose code that people type into something that computers can work with. See ASM.

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 and is typically found running on Windows and Linux. C++ programers will never admit to using Python.

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.

errors Bugs that need to be squashed.

FORTRAN A computer programming language used to play Star Trek games in the 1970s. It has survived death-blows from Python over the years.

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

GitHub It's where we will all be uploaded someday.

i7 A CPU for making your Python code run as fast as C code.

IDE An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. The most popular one right now is Microsoft Visual Studio Code.

import The import command in Python is the same as the load command in BASIC09. It's one of the most over-used commands in Python. Python can import code from the future.

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.

interpreter All of the best programming languages are interpreted. People get things done quicker when they don't have to compile. Less interruptions.

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.

NPC A game character that wants to kill a player character.

Pascal A programming language that Python helped kill.

pip A package installer for Python. It will install packages from the Interweb (a.k.a. iCloud). It's the way most people install Python modules uploaded by other people these days.

print() Always remember to use (and) when using a print function. Them's the rules now.

PyDiceroll A Python module available from this GitHub repository.

PyQt The best GUI for Python. It is the Python version of Qt, which is the C version of the GUI.

Python 3.9+ A version of Python, used the-world-over, that PyDiceroll was written for.

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.

RNG Random number generator. Mostly used as a meme these days. But still has its very practical uses.

RPG Role-playing games use dice. PyDiceroll makes attempts at rolling the dice for the players and for the NPCs.

Sphinx The Python software used to publish this much-needed 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

ELEVEN

OPEN SOURCE

11.1 MIT License

LICENSE AGREEMENT

11.2 Contact

Questions? Please contact shawndriscoll@hotmail.com

CHAPTER

TWELVE

FFE AGREEMENT

CHAPTER

THIRTEEN

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

FOURTEEN

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

p
PyDiceroll, 27

INDEX

Numbers

80286, 29

A

ASM, 29

B

BASIC09, 29
Bugs, 29
built-in function
    roll(), 9

C

C, 29
CMD, 29
COBOL, 29
compiler, 29
concatenation, 29
CPython, 29

D

D100, 29
debug, 29
dice, 29

E

errors, 29

F

FORTRAN, 29

G

game, 29
GitHub, 29

I

i7, 29
IDE, 29
import, 30
integer, 30
interpreter, 30

L

log, 30

M

module, 30
    PyDiceroll, 27

N

no dice, 30
NPC, 30

P

Pascal, 30
pip, 30
print(), 30
PyDiceroll, 30
    module, 27
PyQt, 30
Python 3.9+, 30

R

rabbit hole, 30
random, 30
RNG, 30
roll()
    built-in function, 9
roll() (*in module PyDiceroll*), 27
RPG, 30

S

Sphinx, 30
string, 30

Y

your own code, 30