# xdice Documentation

***Release 1.2.1***

**Olivier Massot**

**Sep 02, 2020**

# Contents

Contents:

CHAPTER 1

# Introduction

## 1.1 Presentation

*xdice* is a dice library for Python that provides the main functionality for managing dice, scores, and dice notation patterns.

DiceRollParser has been tested with python 3.4+. *xdice* is under GNU License

To install:

```
pip install xdice
```

## 1.2 What can it do?

- Parse most of common dice notations: '1d6+1', 'd20', '3d%', '1d20//2 - 2*(6d6+2)', 'max(1d4+1,1d6)', '3D6L2', 'R3(1d6+1)'... etc.
- Manipulate Dice, Pattern, and Score as objects.
- Roll trough command-line or API
- Understand any mathematical expression

### 1.2.1 Examples

```
import dice

score = dice.roll("2d6+18")

print(score)
>> 28
```

(continues on next page)

(continued from previous page)

```
print(score*2)
>> 56
print(score.format())
>> '[5,6]+18'

score = dice.roll("6D%L2")

print(ps, ps.format(verbose=True))
>> 315      '6D%L2(scores:[80, 70, 76, 89], dropped:[2, 49])'
```

CHAPTER 2

# Dice Notation

*Dice notation* is nearly fully understood by pydice.

*xdice* is case insensitive.

## 2.1 Dice

> Patterns describes here can be passed to the Dice.parse() class method, and will then return the corresponding Dice object.

See Wikipedia for a complete definition.

### 2.1.1 Bases

Die rolls are given in the form AdX. A (amount) and X (sides) are variables, separated by the letter "d", which stands for die or dice.

- A is the number of dice to be rolled (1 if omitted).
- X is the number of faces of each die.

  > For example, if a game would call for a roll of d4 or 1d4 this would mean, "roll one 4-sided die." 3d6 would mean, "roll three six-sided dice"

### 2.1.2 Default values

If the A value is omitted, it is assumed to be a 1.

If the X value is omitted, it is assumed to be a 20. This behavior can be modified trough the class property Dice.DEFAULT_SIDES.

### 2.1.3 D% Notation

The `D%` notation is allowed, and read as `D100`.

### 2.1.4 Selective results

The `AdX` pattern can be followed by `Ln` and/or `Hn` ('L' and 'H' respectively stand for lowest and highest).

In this case, the lowest/highest n scores will be discard when the dice will be rolled.

> Eg: `3D6L1` will roll three 6-sided dice, and drop the lowest, while `3D6H1` will roll three 6-sided dice, and drop the highest.

Notes:

- If no number follow the 'L' or 'H', it is assumed to be a 1.
- 'L' and 'H' can be combined inside a single pattern, but 'L' must precede 'H': `6D6L1H2`

### 2.1.5 Exploding dice

Append an `X` or a `!` to a pattern to make the dice 'explode'. 'Explode' means each maximal score will trigger a new roll. The resulting score will be add to the results.

For example, if `3d6!` give `[6,3,2]`, one more die will be rolled (because 6 is the max value) The final result could be `[6,3,2,4]`.

### 2.1.6 Fudge dice

Use the `XdF` notation to use fudge dice.

## 2.2 Patterns

Patterns describes here can be passed to the Pattern.parse() class method.

`AdX` notations can be used in more complex expressions.

Any mathematical expression is allowed:

```
>> 1d10+1d5+1
>> 1d20-6
>> 1d6*2
>> 2d20//4
>> 1d6*(1d4**2)
```

Following builtin python functions are also allowed: `abs`, `max`, `min`. That mean you can parse patterns like `max(1d6+1, 2d4)`.

### 2.2.1 Repeat

The `Rn(AdX)` notation can be used to roll n times the `AdX` command.

For example, the pattern `R3(2d6+2)` will roll `2d6+2` three times: `(2d6+2)+(2d6+2)+(2d6+2)`

## 2.3 Examples

- `1d6` > Roll a 6-sided die
- `1d6+3` > Roll a 6-sided die, then add 3
- `2*(1d6+3)` > Roll a 6-sided die, add 3, then multiply by 2
- `3d6L2` > Roll three 6-sided dice, and drop the two lowest.
- `R2(1d6+3)` > Similar to `1d6+3+1d6+3`
- `1d%` > Similar to `1d100`
- `d6` > Similar to `1d6`
- `min(1d6+10,3d6)` > Keep the minimal score between `1d6+10` and `3d6`

# CHAPTER 3

# Command-Line

Run `python roll.py [options] <expr>`

```
usage: roll [-h] [-V] [-n] [-v] expression [expression ...]

Command Line Interface for the xdice library

positional arguments:
  expression      mathematical expression(s) containing dice <n>d<s> patterns

optional arguments:
  -h, --help      show this help message and exit
  -V, --version   print the xdice version string and exit
  -n, --num_only  print numeric result only
  -v, --verbose   print a verbose result
```

- Basic use

```
python roll.py 1d6+1
>> 4       ([3]+1)
```

- Multiple expressions

```
python roll.py 1d6+1 2d8
>> 6       ([5]+1)
>> 9       ([3, 6])
```

- Numeric score only (-n)

```
python roll.py -n 1d6+1
>> 2
```

- Verbose (-v)

```
python roll.py -v 2*(3D6L1+2D4)+R3(1d4+2)
>> 32       (2*(3d6l1(scores:[1, 3], dropped:[1])+2d4(scores:[1,␣
↪4]))+(1d4(scores:[2])+2+1d4(scores:[2])+2+1d4(scores:[4])+2))
```

CHAPTER 4

# API Reference

Import the *xdice* library with *import dice*

## 4.1 The dice module

### 4.1.1 dice.compile(pattern_string)

Similar to *xdice.Pattern(pattern_string).compile()*

### 4.1.2 dice.roll(pattern_string)

Similar to *xdice.Pattern(pattern_string).roll()*

### 4.1.3 dice.rolldice(faces, amount=1, drop_lowest=0, drop_highest=0)

Similar to *xdice.Dice(faces, amount, drop_lowest, drop_highest).roll()*

## 4.2 Dice class

Set of dice.

### 4.2.1 Dice.__init__ (sides, amount=1, drop_lowest=0, drop_highest=0)

Instantiate a set of dice.

### 4.2.2 Properties

- *dice.sides*: number of sides of the dice
- *dice.amount*: amount of dice to roll
- *dice.drop_lowest*: amount of lowest scores to drop
- *dice.drop_highest*: amount of highest scores to drop
- *dice.name* : Descriptive name of the Dice object

### 4.2.3 dice.roll()

Role the dice and return a Score object

### 4.2.4 *[classmethod]* Dice.parse(cls, pattern)

Parse a pattern of the form 'AdX', where A and X are positive integers, then return the corresponding Dice object. Use 'AdX[Ln][Hn]' to drop the n lowest and/or highest dice when rolled.

## 4.3 Score class

Score is a subclass of integer, you can then manipulate it as you would do with an integer.

It also provides an access to the detailed score with the property 'detail'.
'detail' is the list of the scores obtained by each dice.

Score class can also be used as an iterable, to walk trough the individual scores.

```
eg:
    >>> s = Score([1,2,3])
    >>> print(s)
    6
    >>> s + 1
    7
    >>> list(s)
    [1,2,3]
```

### 4.3.1 Score.__new__(iterable, dropped=[], name='')

*iterable* should only contain integers

Score value will be the sum of the list's values.

### 4.3.2 Properties

- *score.detail*: similar to list(score), return the list of the individual results
- *score.name*: descriptive name of the dice rolled

- *score.dropped*: list of the dropped results

### 4.3.3 score.format(verbose=False)

A formatted string describing the detailed result.

## 4.4 Pattern class

Dice notation pattern.

### 4.4.1 Pattern.__init__ (instr)

Instantiate a Pattern object.

### 4.4.2 pattern.compile()

Parse the pattern. Two properties are updated at this time:

- *pattern.format_string*

  The ready-to-be-formatted string built from the `instr` argument.

  *Eg: '1d6+4+1d4' => '{0}+4-{1}'*

- *pattern.dices*

  The list of parsed dice.

  *Eg: '1d6+4+1d4' => [(Dice; sides=6;amount=1), (Dice; sides=4;amount=1)]*

### 4.4.3 pattern.roll()

Compile the pattern if it has not been yet, then roll the dice.

Return a PatternScore object.

## 4.5 PatternScore class

PatternScore is a subclass of **integer**, you can then manipulate it as you would do with an integer.

Moreover, you can get the list of the scores with the score(i) or scores() methods, and retrieve a formatted result with the format() method.

### 4.5.1 pattern_score.scores()

Returns the list of Score objects extracted from the pattern and rolled.

### 4.5.2 pattern_score.format(verbose=False)

A formatted string describing the detailed result.