
vedicpy Documentation

Release 0.1.0

Utkarsh Mishra

Sep 18, 2020

Getting started

1	Installation	3
2	Tutorial	5
3	Troubleshooting	7
4	Compliment	9
5	Cube	11
6	Cuberoot	15
7	Divisibility	17
8	Division	19
9	Multiplication	21
10	Recurring	31
11	Square	33
12	Squareroot	37
13	BSD 3-Clause License	41
14	Help and Contact	43
15	Contributing	45
16	Credits	47



A Python Package for Vedic Mathematics

For humans, through regular mathematical steps, solving problems sometimes are complex and time-consuming. But using Vedic Mathematic's General Techniques (applicable to all sets of given data) and Specific Techniques (applicable to specific sets of given data), numerical calculations can be done very fast.

This package is a python implementation of Vedic mathematical sutras. It uses the Vedic mathematics for performing basic mathematical operations like multiplication, division, square roots, cube roots etc.

Since Vedic maths sutras work on individual digits in a number as opposed to the whole number, the implementation works slower on small digit numbers but works faster on larger digit numbers and some other operations like finding the square root or the cube root of a number.

CHAPTER 1

Installation

The simplest way to install `vedicpy` is through the Python Package Index (PyPI). This will ensure that all required dependencies are fulfilled. This can be achieved by executing the following command:

```
pip install vedicpy
```


This section covers the fundamentals of developing with **vedicpy**, including a package overview, basic and advanced usage.

2.1 Overview

The *vedicpy* package is structured as collection of submodules:

- *vedicpy*
 - *vedicpy.compliment* Functions for calculating the compliment of a number.
 - *vedicpy.cube* Functions for calculating cube of a number.
 - *vedicpy.cuberoot* Functions for checking and calculating cube root of a number.
 - *vedicpy.divisibility* Function for finding whether a number is divisible by the given number or not.
 - *vedicpy.division* Function for calculating quotient and reminder.
 - *vedicpy.multiply* Functions for calculating the multiplication of two number using vedic mathematical sutras.
 - *vedicpy.recurring* Function for converting fractional number to its corresponding recurring decimal.
 - *vedicpy.square* Functions for calculating square of a number.
 - *vedicpy.squareroot* Functions for checking and calculating square root of a number.

2.2 Quickstart

Before diving into the details, we'll walk through a brief example program

```
# Example of calculating the cube of a number
import vedicpy as vedic

# calling cube_2digit_number from vedic.cube
result = vedic.cube.cube_2digit_number(67)

print(result)
```

In the program we first call the package by using `import` and by giving a compact syntax to it by using `vedic` as the name.

Then we simply call the `cube_2digit_number` function from `cube` module present in `vedicpy`.

As the name suggest `cube_2digit_number` function only cubes 2 digit integer numbers and returns an interger value that is stored in variable `result` and then we simply print that value of the variable.

CHAPTER 3

Troubleshooting

If you have questions about how to use `vedicpy`, please email me at utkarsh.um07@gmail.com.

For bug reports and other, more technical issues, consult the [github issues](#).

3.1 Important Error

Vedic Mathematics doesn't provide a way to calculate square root and cube root accurately. So, if it says that the number is a perfect square or a perfect cube there is still some chance that it is not.

4.1 1) compliment_to_power_of10

The Complement of a number is the difference between that number and the next higher power of 10. 3 is the complement of 7 (as next higher power of 7 is 10). 34 is the complement of 66 (as next higher power of 66 is 100).

Vedic Sutra:

Nikhilam Navatah Charamam Dasatah

which means, All from 9 and last from 10.

Details: We have to get the complement (Nikhilam) for the entire number by using 10 for the digit in the units place and by using 9 for the remaining digits.

Implementation:

```
import vedicpy as vedic

a= vedic.compliment.compliment_to_power_of10(123)
print(a)
```

```
>>> 877
```


5.1 1) cube_a_number_near_powerof10

Yavadunam Method

This method explained earlier can be used in this case also, with modifications. The answer consists of 3 portions as given here under.

- Twice the excess (deviation) is added to the number. This forms Left Portion of the answer.
- The product of new excess and the original excess forms the Middle Portion of the answer.
- The cube of the initial excess forms the Right Portion of the answer.

This is explained below with examples.

Example 1: Find the cube of 13.

Base = 10. Deviation = +3.

New excess = 3 + 6 = 9

(Deviation + Twice deviation) Hence,

$$\begin{array}{r}
 13^3 \\
 \hline
 13+6 : 9 \times 3 : 3^3 \\
 \hline
 19 : 27 : 27 \\
 \hline
 1 \ 9 \ 7 \ 7 \\
 \hline
 2 \ 2 \\
 \hline
 2 \ 1 \ 9 \ 7 \\
 \hline
 \end{array}$$

Therefore, $13^3 = 2197$

Example 2: Find the cube of 106.

Base = 100. Deviation = +6.

New excess = 6 + 12 = 18

(Deviation + Twice deviation) Hence,

$$\begin{array}{r}
 106^3 \\
 \hline
 106+12 : 18 \times 6 : 6^3 \\
 \hline
 118 : 108 : 216 \\
 \hline
 1 \ 1 \ 8 \ 0 \ 8 \ 1 \ 6 \\
 \hline
 1 \ 2 \\
 \hline
 1 \ 1 \ 9 \ 1 \ 0 \ 1 \ 6 \\
 \hline
 \end{array}$$

Therefore, $106^3 = 1191016$

Implementation:

```
import vedicpy as vedic

a= vedic.cube.cube_a_number_near_powerof10(103)
print(a)
```

```
>>> 1092727
```

5.2 2) cube_2digit_number

Straight Cubing of 2 digit numbers

To find cube of any number directly we use the formula: $(a + b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$

We rewrite this as

$$\begin{array}{rcccc} a^3 & a^2b & ab^2 & b^3 \\ & 2a^2b & 2ab^2 & \\ \hline a^3 & 3a^2b & 3ab^2 & b^3 \end{array}$$

The form makes it easy to compute the cube any 2 digit number. The following examples will show how this could be done.

Method

- Find the values of a^3 , a^2b , ab^2 , b^3 and write them as shown.
- Also double the vales of a^2b , ab^2 and write them under respective column.
- Compute the cube of the number from the result.

Example 4: Find the cube of 47.

Here, $a = 4$, $b = 7$.

$$\begin{array}{r}
 64 \ 112 \ 196 \ 343 \\
 224 \ 392 \\
 \hline
 64 \ 336 \ 588 \ 343 \\
 \hline
 6 \ 4 \ 6 \ 8 \ 3 \\
 3 \ 8 \ 4 \\
 3 \ 5 \ 3 \\
 \hline
 1 \ 0 \ 3 \ 8 \ 2 \ 3 \\
 \hline
 \hline
 \text{Hence, } 47^3 = 103823
 \end{array}$$

Implementation:

```
import vedicpy as vedic

a= vedic.cube.cube_2digit_number(37)
print(a)
```

```
>>> 50653
```


6.1 1) cuberoot_check

Let's define something called a **Digital root**.

It is the sum obtained after iteratively adding the digits of a number, till a single digit remains.

For example,

- For 345, digital root of 345 $\Rightarrow 3 + 4 + 5 = 12$. Now, $12 \Rightarrow 1 + 2 = 3$. $12 = 1 + 2 = 3$. So, digital root of 345 = 3.
- For 12345678, digital root is $1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 = 36$. Now, $3 + 6 = 9$. So, digital root of 12345678 = 9.

Turns out that for all perfect cubes, the digital root will either be 1, 8, 9. 0 is not included as 0 is a perfect cube of itself.

Anyways, if for a number xx you get a digital root that is not 1, 8, 9 you can confidently say that xx is NOT a perfect cube.

If the digital root is 1, 8, 9, 0 the number may or may not be a perfect cube.

Implementation:

```
import vedicpy as vedic

a= vedic.cuberoot.cuberoot_check(123)
print(a)
print(type(a))
```

```
>>> False
>>> <class 'bool'>
```

This function returns a `boolean` value.

6.2 2) cuberoot_under_1000000

Example 2:

Find the cube root of 195112.

No of cube root digits = $6/3 = 2$.

Grouping of digits = 195 ' 112

First cube root digit = 5 ($5^3 < 195$)

Modified Divisor = $3 \times 5^2 = 75$.

$$\begin{array}{r} 75 \overline{) 195 \ 1 \ 1 \ 2} \\ \underline{70 \ 101 \ 51} \\ 58.00 \end{array}$$

Working (Mental)

Step 1: $195 - 5^3 = 195 - 125 = 70$. As shown

Step 2: $701 \div 75 = 8 \text{ R } 101$. As shown

Step 3: But, $3ab^2 = 3 \times 5 \times 8^2 = 960$

$$1101 - 960 = 141 \div 75 = 1 \text{ R } 66$$

Step 4: But, $b^3 = 8^3 = 512$

$$512 - 512 = 0 \div 75 = 0 \text{ R } 0$$

Hence, Cube Root of 195112 = 58.

Implementation:

```
import vedicpy as vedic

a= vedic.cuberoot.cuberoot_under_1000000(175616)
print(a)
```

```
>>> 56
```

Vedic Mathematics doesn't provide a way to cube root accurately. So, if it says that the number is a perfect cube there is still some chance that it is not.

7.1 1) divisibility_under10

If we divide one number by another number and get a whole number, we say that the first number is divisible by the second. This property of division is called divisibility. For example,

25 is divisible by 5.; 12 is divisible by 2, 3, 4, 6.

63 is divisible by 3, 7, 9. etc.

Divisibility Criteria - Here are divisibility Criteria for a first few integers:

Divisor	Criteria	Examples
2	All even numbers	12; 24; 136
3	Sum of digits is divisible by 3	15; 234
4	Last two digits divisible by 4	1932; 2016
5	Last digit is 0 or 5	15; 210; 305
6	Numbers divisible by 2 and 3	36; 1236
8	Last four digits divisible by 8	451936
9	Sum of digits divisible by 9	27; 171;

Implementation:

```
import vedicpy as vedic

# divisibility_under10() function takes two arguments,
# first one is dividend and the other one is divisor
vedic.divisibility.divisibility_under10(108, 9)
```

```
>>> The number is divisible.
```

The function doesn't return any value.

The divisibility test is only applicable for divisor less than 10 excluding 1 and 7.

8.1 1) division_by9

Division by 9

$$32 \div 9$$

$$9 \overline{) \textcircled{3} 2}$$

$$\textcircled{3} \text{ r} 5 \quad \text{where } 5 = 3 + 2$$

$$52 \div 9$$

$$9 \overline{) \textcircled{5} 2}$$

$$\textcircled{5} \text{ r} 7 \quad \text{where } 7 = 5 + 2$$

$$75 \div 9$$


$$9 \overline{) \textcircled{7} 5}$$

$$\textcircled{7} \text{ r} 12 \quad \text{where } 12 = 7 + 5 \quad \text{remainder} > 9$$

$$= 8 \text{ r} 3$$

$$3102 \div 9$$

$$9 \overline{) 3102}$$



$$312 \div 9$$

$$9 \overline{) 312}$$

$$34 \text{ r} 6$$

When dividing by 9,
The **remainder** is always the
digit sum of the original number

Implementation:

```
import vedicpy as vedic

# division_by9() function takes a single argument that is dividant.
vedic.division.division_by9(110)
```

```
>>> The quotient is: 12
>>> The reminder is: 2
```

The function doesn't return any value.

9.1 1) multiply_by_9group

Example 1: Multiply 4×9

$$\begin{array}{r} 4 \times 9 \\ \hline 4 - 1 / 6 \\ \hline 36 \end{array}$$

Thus, $4 \times 9 = 36$

Example 2: Multiply 76×99

$$\begin{array}{r} 76 \times 99 \\ \hline 76 - 1 / 24 \\ \hline 7524 \end{array}$$

Thus, $76 \times 99 = 7524$

Example 3: Multiply 353×999

$$\begin{array}{r} 353 \times 999 \\ \hline 353 - 1 / 647 \\ \hline 352647 \end{array}$$

Thus, $353 \times 999 = 352647$

Working from left to right (Mental work)

LHS = $4 - 1 = 3$ (Ekanyunena)

RHS = $10 - 4 = 6$ (Complement of 4 from 10)

Working from left to right (Mental work)

LHS = $76 - 1 = 75$ (Ekanyunena)

RHS = $100 - 76 = 24$ (Complement of 76 from 100)

Working from left to right (Mental work)

LHS = $353 - 1 = 352$ (Ekanyunena)

RHS = $1000 - 353 = 647$ (Complement of 353 from 1000)

Implementation:

```
import vedicpy as vedic

a= vedic.multiply.multiply_by_9group(234)
print(a)
```

```
>>> 233766
```

9.2 2) multiply_base_near_powerof10

Nikhilam Multiplication

As the deviation is obtained by Nikhilam sutra we call the method as Nikhilam multiplication. This is a special method to multiply two numbers near a base or one number near the base and the other a little away from the base. The method of multiplication is given below.

Method

- Write the numbers one below the other in base system.
- Divide the answer space into LHS and RHS by placing a slash (/) or a colon (:).
- Add or subtract one number with the deviation of the other number and write it on the LHS. (i.e., cross-sum or cross-difference)
- Write the product of the deviations on RHS.
- The number of digits on RHS must be same as the number of zeros in the base. If less, prefix the answer with the zeros. If more, transfer extra digits to the RHS. (Digit Rule)
- Take due care to the sign (+/-) while adding or multiplying the numbers.
- Remove the slash.

Three different cases are possible.

Example 1: Multiply 6 x 8.

$$\begin{array}{r}
 6 - 4 \\
 \times 8 - 2 \\
 \hline
 4 / 8 \\
 \hline
 48
 \end{array}$$

Thus, $6 \times 8 = 48$

Working from left to right (Mental work)

Base = 10, RHS Digits = 1

LHS = $6 - 2 = 4$ or $8 - 4 = 4$

RHS = $-4 \times -2 = 8$

Note: Numbers are written in Base System.

Example 2: Multiply 92 x 97.

$$\begin{array}{r}
 92 - 08 \\
 \times 97 - 03 \\
 \hline
 89 / 24 \\
 \hline
 8924
 \end{array}$$

Thus, $92 \times 97 = 8924$

Working from left to right (Mental work)

Base = 100, RHS Digits = 2

LHS = $92 - 03 = 89$ or $97 - 08 = 89$

RHS = $-08 \times -03 = 24$

Note the number of digits in deviation.

Example 3: Multiply 91 x 99.

$$\begin{array}{r}
 91 - 09 \\
 \times 99 - 01 \\
 \hline
 90 / 09 \\
 \hline
 9009
 \end{array}$$

Thus, $91 \times 99 = 9009$

Working from left to right (Mental work)

Base = 100, RHS Digits = 2

LHS = $91 - 01 = 90$ or $99 - 09 = 90$

RHS = $-09 \times -01 = 09$ (Digit Rule)

Implementation:

```
import vedicpy as vedic

a= vedic.multiply.multiply_base_near_powerof10(109,91)
print(a)
```

```
>>> 9919
```

9.3 3) multiply_equdigit_number

Multiplication of 2 x 2 digit numbers.

Method

- Write the numbers one below the other.
$$\begin{array}{r} a \ b \\ \times c \ d \\ \hline \end{array}$$
- Divide the answer space into three parts using slash (/) or colon (:).
- Step 1: Find $(a \times c)$ – Multiplying vertically on left side.
- Step 2: Find $(a \times d + b \times c)$ – Multiplying crosswise and adding.
- Step 3: Find $(b \times d)$ – Multiplying vertically on right side.
- Write the respective products at appropriate places in the answer space.

Multiplication of 3 x 3 digit numbers.

Method

- Write the numbers one below the other.
$$\begin{array}{r} a \ b \ c \\ \times d \ e \ f \\ \hline \end{array}$$
- Divide the answer space into 5 parts using slash (/) or colon (:).
- Step 1: Find $(a \times d)$
- Step 2: Find $(a \times e + b \times d)$
- Step 3: Find $(a \times f + b \times e + c \times d)$.
- Step 4: Find $(b \times f + c \times e)$
- Step 5: find $(c \times f)$
- Write the respective products at appropriate places in the answer space.

Multiplication of 4 x 4 digit numbers.

Method

- Write the numbers one below the other.

$$\begin{array}{r} a \ b \ c \ d \\ \times e \ f \ g \ h \\ \hline \end{array}$$
- Divide the answer space into 7 parts using slash (/) or colon (:).
- Step 1: Find (a x e)
- Step 2: Find (a x f + b x e)
- Step 3: Find (a x g + b x f + c x e).
- Step 4: Find (a x h + d x e + b x g + c x f)
- Step 5: Find (b x h + c x g + d x f)
- Step 6: Find (c x h + d x g)
- Step 7: Find (d x h)
- Write the respective products at appropriate places in the answer space.

Implementation:

```
import vedicpy as vedic

a= vedic.multiply.multiply_equdigit_number(1234, 4567)
print(a)
```

```
>>> 5635678
```

9.4 4) multiply_lastdigit_sumto10

Ekadhikena Multiplication

This is another simple method. The Vedic Sutra used in this method is "One more than the previous one" – Ekadhikena. Two different cases arise here.

Case I – Last digits adding to 10.

The numbers used in this method must obey the following conditions.

Both the numbers must have the same previous digit(s).

The sum of the last digits must be 10.

Numbers like 54 and 56, 42 and 48, 23 and 27, 34 and 36 form the examples.

Example 1:**Multiply 51 x 59**

Same previous digit: 5,

Sum of last digits: 1+9=10.

$$\begin{array}{r} 51 \times 59 \\ \hline 5 \times (5+1) / 1 \times 9 \\ \hline 30 / 09 \end{array}$$

Thus, 51x59=3009.

Note that 0 has been added on RHS.

Example 2:**Multiply 66 x 64**

Same previous digit: 6,

Sum of last digits: 6+4=10.

$$\begin{array}{r} 66 \times 64 \\ \hline 6 \times (6+1) / 6 \times 4 \\ \hline 42 / 24 \end{array}$$

Thus, 66 x 64=4224.

Example 3:**Multiply 123 x 127**

Same previous digits: 12,

Sum of last digits: 3+7=10.

$$\begin{array}{r} 123 \times 127 \\ \hline 12 \times (12+1) / 3 \times 7 \\ \hline 156 / 21 \end{array}$$

Thus, 123 x 127=15621.

Implementation:

```
import vedicpy as vedic

a= vedic.multiply.multiply_lastdigit_sumto10(24, 26)
print(a)
```

```
>>> 624
```

9.5 5) multiply_by11

Multiplication by 11

Multiplication of a number by 11 is very easy. It is as good as addition. This method is explained below.

Method

- Sandwich the given number between zeros.
- Starting from left end add the digits taking them in pairs.
- If the total exceeds 9, retain the first digit and carry over the other digits to the left.

Example1: Multiply 135 x 11

Write the multiplicand as shown
and add the digits in pair.

$$\begin{array}{r} 135 \times 11 \\ \hline 01350 \\ 1485 \\ \hline \end{array}$$

Thus, $135 \times 11 = 1485$

Example 2: Multiply

58403 x 11

$$\begin{array}{r} 58403 \times 11 \\ \hline 0584030 \\ 532433 \\ 11 \\ \hline 642433 \end{array}$$

Thus, $58403 \times 11 = 642433$

Working from left to right (Mental work)

Addition of digits in pair is shown below.

$$0 + 1 = 1,$$

$$1 + 3 = 4,$$

$$3 + 5 = 8,$$

$$5 + 0 = 5$$

Working from left to right (Mental work)

Addition of digits in pair is shown below.

$$0 + 5 = 5, 5 + 8 = 13, 8 + 4 = 12,$$

$$4 + 0 = 4, 0 + 3 = 3, 3 + 0 = 3.$$

Implementation:

```
import vedicpy as vedic
a= vedic.multiply.multiply_by11(103)
print(a)
```

```
>>> 1133
```

9.6 6) multiply_by12

Multiplication by 12

This is similar to the one discussed earlier. But, there is a slight difference. This employs the Vedic Sutra "*Ultimate and twice the penultimate*". According to this, we must add twice the penultimate digit to the ultimate digit.

Consider the number 32. Here, penultimate digit is 3 and the ultimate digit is 2. By the above Sutra, the required sum = $3 \times 2 + 2 = 8$.

Example 1: Multiply 123×12

$$\begin{array}{r} 123 \times 12 \\ \hline 01230 \\ \hline 1476 \\ \hline \end{array}$$

Thus, $123 \times 12 = 1476$

Working from left to right (Mental work)

Write the multiplicand as shown and add twice the penultimate digit to ultimate digit.

$$0 \times 2 + 1 = 1, 1 \times 2 + 2 = 4,$$

$$2 \times 2 + 3 = 7, 3 \times 2 + 0 = 6.$$

Example 2: Multiply 396×12

$$\begin{array}{r} 396 \times 12 \\ \hline 03960 \\ \hline 3542 \\ \hline 121 \\ \hline 4752 \\ \hline \end{array}$$

Thus, $396 \times 12 = 4752$

Working from left to right (Mental work)

Write the multiplicand as shown and add twice the penultimate digit to ultimate digit.

$$0 \times 2 + 3 = 3, 3 \times 2 + 9 = 15,$$

$$9 \times 2 + 6 = 24, 6 \times 2 + 0 = 12.$$

Implementation:

```
import vedicpy as vedic

a= vedic.multiply.multiply_by12(103)
print(a)
```

```
>>> 1236
```

This method can be extended for multiplication with 13, 14, 15, ... , 19 with little modification. Instead of *twice* we have to take *three times*, *four times*, etc. Try this!

9.7 7) multiply_by13

Implementation:

```
import vedicpy as vedic

a= vedic.multiply.multiply_by13(103)
print(a)
```



```
>>> 1339
```

9.8 8) multiply_by14

Implementation:

```
import vedicpy as vedic

a= vedic.multiply.multiply_by14(103)
print(a)
```

```
>>> 1442
```

9.9 9) multiply_by15

Implementation:

```
import vedicpy as vedic

a= vedic.multiply.multiply_by15(103)
print(a)
```

```
>>> 1545
```

9.10 10) multiply_by16

Implementation:

```
import vedicpy as vedic

a= vedic.multiply.multiply_by16(103)
print(a)
```

```
>>> 1648
```

9.11 11) multiply_by17

Implementation:

```
import vedicpy as vedic

a= vedic.multiply.multiply_by17(103)
print(a)
```

```
>>> 1751
```

9.12 12) multiply_by18

Implementation:

```
import vedicpy as vedic

a= vedic.multiply.multiply_by18(103)
print(a)
```

```
>>> 1854
```

9.13 13) multiply_by19

Implementation:

```
import vedicpy as vedic

a= vedic.multiply.multiply_by19(103)
print(a)
```

```
>>> 1957
```

10.1 1) recuring_fractionto_decimal

A decimal with a sequence of digits that repeats itself indefinitely is called recurring decimal.

The Vedic Sutra Ekadhika helps us to convert Vulgar fractions of the type $1/p9$, where $p = 1, 2, 3, \dots$ 9, into recurring decimals. The number of decimal places before repetition is the difference of numerator and denominator. The devisor can be found using Sutra "One more than the previous one".

Conversion of a few vulgar fractions is shown below. The method is simple and does not require actual division. It is only simple division with small figures.

Convert 11/19 into recurring decimal.

The denominator = 19. Previous digit = 1. Ekadhika is $1+1=2$.

$$11/19 = 11/20 = 1.1/2 = 0.\underset{1}{5}\underset{1}{7}\underset{1}{89}\underset{1}{47}\underset{1}{3}\underset{1}{68421}$$

Therefore, $11/19 = 578947368421$.

Convert 1/7 into recurring decimal.

Here, number of decimal places is $(7 - 1) = 6$.

$1/7 = 7/49$. The denominator is 49.

Previous digit = 4. Ekadhika is $4+1=5$.

$$7 / 49 = 7/50 = 0.7/5 = 0.\underset{2}{1}\underset{1}{4}\underset{2}{2}\underset{8}{3}57$$

Therefore, $1/7 = 0.142857$.

Convert 9/39 into recurring decimal.

The denominator = 39. Previous digit = 3. Ekadhika is $3+1=4$.

$$9/39 = 9/40 = 0.9/4 = 0.\underset{1}{2}\underset{3}{3}0769. \text{ Therefore, } 9/39 = 0.230769.$$

Implementation:

```
import vedicpy as vedic

result = vedic.recurring.recuring_fractionto_decimal(11, 19)
print(result)
```

```
>>> 0.578947
```

The functions returns a decimal value with a round off on 6 digits.

11.1 1) square_ending5

Ekadhika method

We studied earlier the method of finding the square of numbers ending in 5 under Ekadhika multiplication. E.g., $35^2 = 3 \times 4 / 25 = 1225$,

$$65^2 = 6 \times 7 / 25 = 4225, \text{ etc.}$$

If the number is large we can use Urdhva Tiryak multiplication in concurrence to achieve this. This is illustrated in the following examples.

Example 1: Find the square of 285.

$$285^2 = 28 \times 29 / 25$$

$$\begin{array}{r} 28 \\ \times 29 \\ \hline 442 \\ 37 \\ \hline 812 \end{array}$$

Therefore, $285^2 = 812/25 = 81225$.

Example 2: Find the square of 1235.

$$1235^2 = 123 \times 124 / 25$$

$$\begin{array}{r} \text{But, } 123 \\ \times 124 \\ \hline 14142 \\ 111 \\ \hline 15252 \end{array}$$

Therefore, $1235^2 = 15252/25 = 1525225$.

Implementation:

```
import vedicpy as vedic

a= vedic.square.square_ending5(35)
print(a)
```

```
>>> 1225
```

11.2 2) square_near_powerof10

Yavadunam Method

The Vedic Sutra "Deviate as much as deviation and add square of the deviation". This is known as Yavadunam Sutra. We shall see the application of this Sutra to find the square of numbers.

The answer consists of 2 portions as given here under.

- The excess (deviation) is added to the number. This forms the Left Portion of the answer.
- The square of the initial excess forms the Right Portion of the answer.

This is explained below with examples.

Example 1: Find the square of 12.

Here, Base = 10, Deviation = $12 - 10 = +2$.

Hence, $12^2 = 12 + 2 / 2^2 = 144$.

Example 2: Find the square of 108.

Here, Base = 100, Deviation = $108 - 100 = +08$.

$108^2 = 108 + 08 / 08^2 = 11664$. (Digit Rule)

Example 3: Find the square of 1015.

Base = 1000, Deviation = $1015 - 1000 = +015$.

$1015^2 = 1015 + 015 / 015^2 = 1030225$.

Implementation:

```
import vedicpy as vedic

a= vedic.square.square_near_powerof10(98)
print(a)
```

```
>>> 9604
```

11.3 3) square_under100

Squaring of 2 digit numbers

We have: $(a + b)^2 = a^2 + 2ab + b^2$

Or $(a + b)^2 = D(a) + D(ab) + D(b)$.

We use this property as follows.

$(ab)^2 = D(a) : D(ab) : D(b)$

This is clear from the following examples.

Example 1: Find the square of 36.

$$\begin{array}{r}
 36^2 \\
 \hline
 D(3):D(36):D(6) \\
 \hline
 9 : 36 : 36 \\
 \hline
 9 \ 6 \ 6 \\
 3 \ 3 \\
 \hline
 1 \ 2 \ 9 \ 6
 \end{array}$$

Working (Mental)

Step 1: $D(3) = 3^2 = 9$

Step 2: $D(36) = 2 \times 3 \times 6 = 36$

Step 3: $D(6) = 6^2 = 36$.

Hence, $36^2 = 1296$

Implementation:

```
import vedicpy as vedic

a= vedic.square.square_under100(69)
print(a)
```

```
>>> 4761
```

11.4 4) square_from100_to1000

Squaring of 3 digit numbers

We have: $(a + b + c)^2 = a^2 + 2ab + (b^2 + 2ac) + 2bc + c^2$

Or $(a+b+c)^2 = D(a)+D(ab)+D(abc)+D(bc)+D(c)$.

We use this property as follows.

$(abc)^2 = D(a) : D(ab) : D(abc) : D(bc) D(c)$

This is clear from the following examples.

Example 1: Find the square of 234.

$$\begin{array}{r} 234^2 \\ \hline D(2):D(23):D(234):D(34):D(4) \\ \hline 4 : 12 : 25 : 24 : 16 \\ \hline 4 \ 2 \ 5 \ 4 \ 6 \\ 1 \ 2 \ 2 \ 1 \\ \hline 5 \ 4 \ 7 \ 5 \ 6 \end{array}$$

Working (Mental)

Step 1: $D(2) = 2^2 = 4$

Step 2: $D(23) = 2 \times 2 \times 3 = 12$

Step 3: $D(234) = 3^2 + 2 \times 2 \times 4 = 25$.

Step 4: $D(34) = 2 \times 3 \times 4 = 24$

Step 5: $D(4) = 4^2 = 16$.

Hence, $234^2 = 1296$

Implementation:

```
import vedicpy as vedic

a= vedic.square.square_from100_to1000(983)
print(a)
```

```
>>> 966289
```


12.1 1) squareroot_check

For all the numbers ending in 1, 4, 5, 6, & 9 and for numbers ending in even zeros, then remove the zeros at the end of the number and apply following tests:

- **Digital roots** are 1, 4, 7 or 9. No number can be a perfect square unless its digital root is 1, 4, 7, or 9. You might already be familiar with **computing digital roots**. (To find digital root of a number, add all its digits. If this sum is more than 9, add the digits of this sum. The single digit obtained at the end is the digital root of the number.)
- If unit digit ends in 5, ten's digit is always 2.
- If it ends in 6, ten's digit is always odd (1, 3, 5, 7, and 9) otherwise it is always even. That is if it ends in 1, 4, and 9 the ten's digit is always even (2, 4, 6, 8, 0).
- If a number is divisible by 4, its square leaves a remainder 0 when divided by 8.
- Square of even number not divisible by 4 leaves remainder 4 while square of an odd number always leaves remainder 1 when divided by 8.
- Total numbers of prime factors of a perfect square are always odd.

If the number passes all the parameter then it **can** be a perfect square.

Implementation:

```
import vedicpy as vedic

a= vedic.squareroot.squareroot_check(144)
print(a)
print(type(a))
```

```
>>> True
>>> <class 'bool'>
```

This function returns a `boolean` value.

12.2 2) perfect_sqrt_under_sqof100

Example 1: Find the square root of 1156.

No of square root digits = $4/2 = 2$.

Grouping of digits = 11'56.

First square root digit = 3 ($3^2 < 11$)

Modified Divisor = $3 \times 2 = 6$.

$$\begin{array}{r} 11 \ 5 \ 6 \\ 6 \overline{) \quad 2 \ 1} \\ \underline{3 \ 4.0} \end{array}$$

Working (Mental)

Step 1: $11 - 3^2 = 11 - 9 = 2$

Step 2: $25 \div 6 = 4 \text{ R } 1$

Step 3: $16 - D(4) = 16 - 4^2 = 0$

Hence, Square Root of 1156 = 34.

Example 2: Find the square root of 6241.

No of square root digits = $4/2 = 2$.

Grouping of digits = 62'41.

First square root digit = 7 ($7^2 < 62$)

Modified Divisor = $7 \times 2 = 14$.

$$\begin{array}{r} 62 \quad 4 \quad 1 \\ 14 \overline{) 13 \quad 8} \\ \underline{7 \quad 9 \quad 0} \end{array}$$

Working (Mental)

Step 1: $62 - 7^2 = 62 - 49 = 13$

Step 2: $134 \div 14 = 9 \text{ R } 8$

Step 3: $81 - D(9) = 81 - 9^2 = 0$

Hence, Square Root of 6241 = 79.

Implementation:

```
import vedicpy as vedic

a= vedic.squareroot.perfect_sqrt_under_sqof100(144)
print(a)
```

```
>>> 12
```

Vedic Mathematics doesn't provide a way to square root accurately. So, if it says that the number is a perfect square there is still some chance that it is not.

CHAPTER 13

BSD 3-Clause License

Copyright (c) 2020, Utkarsh Mishra All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER 14

Help and Contact

Questions or Trouble related to package? Please contact utkarsh.um07@gmail.com.

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. You can contribute in many ways:

15.1 Types of Contributions

15.1.1 Report Bugs

Report bugs at <https://github.com/utkarsh0702/vedicpy/issues>.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

15.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

15.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

15.1.4 Write Documentation

vedicpy could always use more documentation, whether as part of the official *vedicpy* docs, in docstrings, or even on the web in blog posts, articles, and such.

15.1.5 Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/utkarsh0702/vedicpy/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

15.2 Get Started!

Ready to contribute? Here's how to set up *vedicpy* for local development.

1. Fork the *vedicpy* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/vedicpy.git
```

3. Install your local copy into a virtualenv. Assuming you have *virtualenvwrapper* installed, this is how you set up your fork for local development:

```
$ mkvirtualenv other
$ cd other/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

15.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in *README.rst*.

CHAPTER 16

Credits

16.1 Contributors

- Utkarsh Mishra
- Ashish Kumar