ABANT IZZET BAYSAL UNIVERSITY

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

SYMBOLIC MATHEMATICS WITH PYTHON

MASTER OF SCIENCE

FATİH KÜRŞAT CANSU

BOLU, AUGUST 2017

ABANT IZZET BAYSAL UNIVERSITY
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF MATHEMATICS

SYMBOLIC MATHEMATICS WITH PYTHON

MASTER OF SCIENCE

FATİH KÜRŞAT CANSU

BOLU, AUGUST 2017

APPROVAL OF THE THESIS

SYMBOLIC MATHEMATICS WITH PYTHON submitted by **Fatih Kürşat CANSU** in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE** in **DEPARTMENT OF MATHEMATICS, ABANT IZZET BAYSAL UNIVERSITY** by,

Examining Committee Members	**Signature**
Supervisor Assoc. Prof. Erol YILMAZ Abant Izzet Baysal University	....
Member Ass. Prof. Özgür BAŞTÜRK Ankara University	....
Member Ass. Prof. Dr. İsmail Uğur TİRYAKİ Abant İzzet Baysal University	....

August 03, 2017

Prof. Dr. Duran KARAKAS ....D. Karakaş........

Director of Graduate School of Natural and Applied Sciences

To Fatma Zeynep, Defne Dora and Berin Bade

DECLARATION

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Fatih Kürşat CANSU, PhDc

ABSTRACT

SYMBOLIC MATHEMATICS WITH PYTHON

MSC THESIS
FATIH KÜRŞAT CANSU
ABANT IZZET BAYSAL UNIVERSITY GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF MATHEMATICS
(SUPERVISOR: ASSOC. PROF. EROL YILMAZ)

BOLU, AUGUST 2017

Python has been a very popular programming language in recent years. SymPy is an open source. Python library which has been developed aiming extensibility, easy usage and accessibility. These characteristics have made SymPy a popular symbolic scientific library in the world of mathematics. In this work, the main aim is presenting the main features of SymPy, giving a detailed description of its features, and a discussion of selected submodules. The solutions to the provided suplementary problems are also going to help in a deep understanding of the deteails of the architecture and the features of SymPy.

KEYWORDS: Python, Symbolic Mathematics, Anaconda, Computer Algebra Systems.

ÖZET

PYTHON İLE SEMBOLİK MATEMATIK UYGULAMALARI
YÜKSEK LISANS TEZI
FATIH KÜRŞAT CANSU
ABANT İZZET BAYSAL ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ
MATEMATIK ANABILIM DALI
(TEZ DANIŞMANI: DOÇ.DR. EROL YILMAZ)

BOLU, AĞUSTOS - 2017

Python programlama dili son yıllarda oldukça popular olmuş bir programlama dilidir. SymPy ise Python dili ile yazılmış kaynak kodları geliştiriciler için açık olan bilgisayar tabanlı cebirsel bir Python kütüphanesidir. Bu yapı oluşturulurken temelde odaklanılan noktalar kolay ulaşılabilir ve kullanılabilir olması, esnek olması ve interaktif bir şekilde kullanılabilmesidir. Bu sayılan özellikler zaten SymPy dilini özellikle bilimsel Python modülleri arasında oldukça popular hale getirmişlerdir. Yapılan bu çalışma SymPy dilinin genel mimarisini, detaylı kullanımını ve özelliklerinin uygulamalarını içermektedir. Ayrıca matematiksel uygulamalar ve örneklerle SymPy dilinin yapısının daha iyi anlaşılması yazılan uygulamalar ile sağlanmaya çalışılmıştır.

ANAHTAR KELİMELER: Python, Sembolik Mathematik, Anaconda, Bilgisayar Temelli Cebir Sistemleri

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

SymPy **:** Symbolic Python
BSD **:** Berkeley Software Distribution
GUI **:** Graphical User Interface

ACKNOWLEDGEMENTS

I would like to express my highly gratitude to my supervisor Assist.Prof. Dr. Erol YILMAZ for his guidance, advice, criticism, encouragements and insight throughout the research.

I would also like to thank Assoc. Prof. Dr. Sibel Kılıçarslan CANSU for her help, advice, hope, suggestions and comments.

I would also like to thank the Informatic Olympiad students of Bahçeşehir Science and Technology High Schooll; and for clever algorithms developed by Alpaslan KARGIN.

1. INTRODUCTION

SymPy is a full featured computer algebra system (CAS) which is built with Python programming language (Lutz, 2013). It is also free, open sourced, improvable and licensed under 3-clause BSD licence (Rosen, 2005). The SymPy initiation project was started in 2005 by Ondrej Certic and the library has been further improved by over 500 contributers all from over the world. The main reason for rapid improvement has been the contributions from the GitHub community (Raymond, 1999). Hundreds and thousands of software developers from all over the world have been using this community as a startup and software bazaar. The community model, accessibility of the code base and easy usage of Python Language made the SymPy a popular computational algebraic system.

Python is a dynamicly typed programming language which is easy to learn and to code with. Due the part this focus, it has become a popular language for scientific computing and data science, with broad ecosystem and libraries (Oliphant, 2007). SymPy is itself also using by another computer based algebra systems such as Sagemath (pure and applied mathematics) (The Sage Developers, 2017), yt (astronomy and astrophysics, package for analyzing and visualizing big-data) (SymPy Developers, 2017), PyDy (multibody dynamics) (Gede et al., 2013), SfePy(finite elements) (Cimrman, 2014), galgebra (geometric algebra), Quameon (quantum monte carlo in python).

When compared with other computational software systems, the SymPy does not invent its own software language. Python itself is used for internal implementation and end user interaction. For example, Sage also is based on Python as its language. But Sage is over a gigabyte and SymPy is lightweight. Besides these, it enables the users and developers to focus on mathematics rather than language design. Python is a well-constructed, bench-tested software language. By reusing an already existing language, end users are able to focus on those things that matter: the mathematics. Especially Python users have an advantage because SymPy version 1.0 officially is compatible with both Pyton 2.x and 3.x versions. Neverthless, Python is an interpreted language after all which makes it, and the packages built on it such as SymPy, a bit slower than compiled programming languages and software packages developed by using them. Hovewer with the use of modern day powerfull computers this disadvantage is overcome easily.

The final important things about SymPy are that it can be used as library and it has no graphical user interface (GUI). Like other Python libraries it can be used with import statements in all Python development environments. As it has been mentioned, there is no built-in GUI in Python; however, SymPy can be integrated to very rich and interactive display systems like Anaconda and Jupyter (Kluyver et al., 2017) frontends, including the Jupyter Notebook and Qt Console. For online systems Anaconda supports an online SymPy interactive environment. Jupyter Notebook and Qt Console also can render SymPy mathematical expressions using Mathjax (Cervone, 2012) or Latex.

All examples in this thesis are based on SymPy version 1.0, Python version 3.6.1, mpmath version 0.19 and Anaconda 4.3.1.; Windows 7 and Windows 10 have been used as operating systems. Operating systems and all software packages used during the writing of the thesis are either originally licensed or open sourced.

2. AIM AND SCOPE OF THE STUDY

The main objective of this thesis is discussing the key components of the SymPy Library in depth with its applications. Section "Algebra and Symbolic Mathematics" introduces symbolic mathematics by using SymPy Library. This section begins with the basic of representing and manipulation of algebraic expressions before more complex matters. Section "Calculus with SymPy" discusses the mathematical functions available under Python standard library and SymPy. Section Fractals and Geometric Shapes discuss patches in matplotlib that allow to construct geometric shapes and fractals. Section "Problems and Solutions" contains over 50 mathematical algorithm problems from "Project Euler" which are solved by the author of this thesis.

The following line imports all functions of SymPy into code block when executed:

*>>> from SymPy import **

All the given examples in this thesis can be tested on Anaconda or SymPy Live which is an online shell that makes use of Google App Engine (Ciuarana, 2009). SymPy Live can be used online at the address http://docs.SymPy.org.

3. MATHEMATICS AND PYTHON

3.1 Algebra and Symbolic Mathematics

Mathematical problems and their solutions have all involved the handling of numbers. But not all mathematical problems are about the number manipulation and calculation. There is another way to be learned, taught and practiced, and that is doing mathematics in terms of symbols. Besides the numbers *x's* and *y's* are also used to calculate to reach solutions in mathematics. We refer the type of mathematics that makes use of only symbolic forms as symbolic math.

3.1.1 Defining Symbols, Symbolic Operations and Basic Assumptions

In symbolic mathematics mathematical operations are done using symbols instead of numbers. This means by using symbols mathematical values and variables are represented in exact form, not approximately. If a variable is not evaluated, then it is left in its symbolic form. In a typical *Python IDLE* we can refer a number by using variables.

The following example shows the difference between an approximate form and a symbolic form. Before the example consider the following statements:

```
>>> x=1
>>> x+x+1
                                        3
```

A label, x, created to refer number 1. Then the statement x+x+1 gives the result 3. What if we want to get the result in terms of x? If we write just x and x+x+1 *Python IDLE* will generate an error message because the variable x is undefined. Python doesn't know what x refer to.

SymPy gives an opportunity to write an expression without referring any integer or any other numerical type. To use a symbol in a code line, we have to create an object of the *Symbol class* like the following:

```
>>> from SymPy import Symbol
>>> x=Symbol('x’)
```

Firstly, the Symbol class has to be imported. The symbol class is already stored in SymPy library. Then the object is created as symbolic. Now we can define an expression in mathematically and we can calculate the result of the operation.

```
>>> from SymPy import symbols
>>> x, y = symbols('x y')
>>> expr = x + 2*y
>>> expr
                            x + 2*y
```

During the thesis the label and the symbol will be named the same because using a non-matching labels and variables can be confusing. For instance, *x=Symbol(‘x’)* so both the variable x and the symbol x has the same name, which is x. We also have an opportunity to change the label name and the variable name as seen in the code below.

```
>>> a=Symbol('x’)
>>> a+a+1
                            2*x+1
```

Besides these, SymPy has a usefull attribute *“.name”.* For example:

```
>>a=Symbol('x’)
>>a.name
                             'x’
```

Instead of writing all symbols separately, all the symbols can be imported in the program in just one line.

```
>>from SymPy import symbols
>>x,y,z=symbols('x y z’)
```

If you want to change the value of any variable, you can use a very practical method.

```
>>> x = symbols('x')
>>> expr = x + 1
>>> expr.subs(x, 2)
                                        3
```

By using the substitution (subs) method, the value of any variable can be changed. If you don’t use this method the symbol ‘x’ will never change the value itself. Also multi-substitution to any expressions can be done.

```
>>> expr = x**3 + 4*x*y - z
>>> expr.subs([(x, 2), (y, 4), (z, 0)])
                                        40
```

SymPy also can do simple addition and multiplication without importing any extra package. Let’s check the interactive screen when expression x*(x+x) as an input.

```
>>p=x*(x+x)
>>p
                                      2*x**2
```

But the expression (x+2)*(x+1) can not be computed by using same way. Because an extra command of SymPy is needed. In SymPy to avoid the mathematical errors like negative square root, some assumptions have to be used. For instance *Symbol(‘t’, positive=True*) will make a symbol named *t* that is assumed to be positive.

```
>>t=Symbol(‘t’, positive=True)
>>sqrt(t**2)
                                        t
```

Some of the basic assumptions are negative, positive, nonpositive, real, integer and prime. All SymPy assumptions can be controlled by *is_assumption*, like *t.is_positive*. In Python there exist three types of Boolean variables; *True*, *False* and *None*. In these cases *None* is generated by Python in case of an unknown value. For example, *Symbol(‘x’, real=True).is_positive* generates *None* because a real symbol can be positive or negative.

3.1.2 Working with Expressions

This is the simple and common way writing a symbolic expression in Python. But a mathematician will need more complicated ways and methods in symbolic mathematics.

3.1.3 Factorizing and Expanding

The *factor()* function factorise a symbolic mathematical expression into its factors. The function *expand()* expands any given expressions as sums. The usage and the flow of these statements are illustrated in the following example. Let's choose the expression as $x^2 + y^2 = (x - y)(x + y)$. Two symbols have to be taken in the expressions and two *Symbol* objects will be created:

```
>>> x,y=symbols('x,y')
>>> from SymPy import factor
>>> expr=x**2-y**2
>>> factor(expr)
                        (x-y)*(x+y)
```

Factorized expressions in a new expression can also be stored by labeling them as a new elements. Let's try this with a more complicated identitiy, $x^3 + 3x^2y + 3xy^2 + y^3 = (x + y)^3$.

```
>>> expr=x**3+3*x**2*y+3*x*y**2+y**3
>>> factors=factor(expr)
>>> factors
```

$$\boldsymbol{(x+y)^3}$$

```
>>> expand(factors)
```

$$\boldsymbol{x^3 + 3x^2y + 3xy^2 + y^3}$$

If you try to factorize any expression which can not be factorized, the original expression will be printed out by SymPy. For instance,

```
>>> expr=x+y+x*y
>>> factor(expr)
```

$$\boldsymbol{xy + x + y}$$

Similarly, if we try to expand any expression which is already expanded, the expand function will return the same expression. Besides these functions, SymPy has more functions to simplfy the expressions.

Table 3.1. Basic simplification functions.

Expand	for expanding polynomials
Factor	for finding factor of polynomial
Collect	for finding coefficeints of a polynomial
Cancel	writing p/q where p and q are in their lowest terms
Trigsimp	for trigonometric function (Fu et all., 2006)
Hyperexpand	for expanding hypergeometric functions (Roach, 1996; Roach, 1997)

3.1.4 Printers and Pretty Printing in SymPy

In Python using string representation is very common because it is readable by Python and a human user. To make the expressions look nicer on paper; *pprint()* function can be used. For a more thorough understanding the difference between the functions *pprint()* and *print()*, is illustrated in the following example:

```
>>> expr=x*x+2*x*y+y*y
>>>expr
                    x*x+2*x*y+y*y
```

In the last example the polynomial expression looks very simple but it is a little bit difficult to figüre out the bases and powers. SymPy also has a two dimensional printing option with *pprint()*. In this option Unicode characters are converted for a better interpretation of mathematical symbols such as square roots, integral signs, paranthesis. But the results of this method can not guarantee good looking print outs without the usage of Latex and *Anaconda*. Now let’s try the same example with using function *pprint()*.

```
>>> from SymPy import pprint
>>> expr=x**2+2*x*y+y**2
>>> pprint(expr)
```

$$\boldsymbol{x^2 + 2 \cdot x \cdot y + y^2}$$

If the aim is having nice look in the outputs, the function *init_printing()* must be used; this will automatically gets the best printer in your environment. By using this function we also avoid the *(asterix) symbols. If the plan is using SymPy interactively and good looking pretty printing, the *init_session()* can be added. This function will automatically import all SymPy functions. So using this command is strongly advised. In all the codes and programs developed in this thesis, *init_session()* function has been used.

```
>>> from SymPy import init_printing
>>> init_printing(order=’rev-lex’)
>>pprint(expr)
```

$$\mathbf{1 + 2 \cdot x + 2 \cdot x^2}$$

In the last example, an extra command *rev-lex* is also used. It is called with *init_printing()*. This shows that, the aim is to print the expression from lower to higher degree terms. Since the live *SymPy Live Shell* used, it is not needed to import *init_printing()* because the line is already imported by the live shell. *Jupyter Notebook* and *Qt Console* users are more fortunate in this regard because both systems use *LaTeX* or *MathJax* for rendering and printing expressions (Perez and Granger, 2007).

The other printing systems such as mathML, str(string), srepr, ASCII pretty printer, Unicode pretty printer and dot are also available in SymPy. As a final example, it will be given a Latex printer which converts a given expression to Latex codes.

```
>>> print(latex(Integral(sqrt(1/x), x)))
\int \sqrt{\frac{1}{x}}\, dx
```

In this thesis, all the codes written in *Jupyter Notebook or Qt Console* first 3 lines are always given as below:

```
from SymPy import *
from SymPy import init_sesion
init_session(quiet=True)
```

Consider the following series,

$$x+\frac{x^2}{2}+\frac{x^3}{3}+\frac{x^4}{4}+\cdots+\frac{x^n}{n}.$$

The aim is to write a program which asks the user to get the maximum power of the expansion. In this series, x is a symbol and n is an integer number which is given by user. So the n'th term will be

$$\frac{x^n}{n}.$$

The series can be printed by using the following codes.

```
'''Print the series:
x+x**2/+x**3/3+x**4/4+...+x**n
'''
from SymPy import *
from SymPy import init_printing, Symbol, pprint

def print_series(n):
  #Initialize printing system in reverse order
  init_printing(order='rev-lex')
  x=Symbol('x')
  series=x
  for i in range(2,n+1):
    series=series+(x**i)/i
  pprint(series)
n=input('Enter the number of the terms you want in the series:
')
print_series(int(n))
```

the out put of the program for n=5 will be

```
Enter the number of the terms you want in the series:  5
     2    3    4    5
    x    x    x    x
x + ── + ── + ── + ──
    2    3    4    5
```

The packages are imported which will be used in the code snippet. Then a function print_series is defined with the variable n. In this function, a line *init_printing(order='rev-lex')* is added because the final polynomial function must be printed in terms of ascending power. In the following section, calculating the sum of the series for an exact value of x is given.

3.1.5 Substituting The Values in an Expression

By now, printing any expression in SymPy is discussed. Now let’s consider how the value of an expression for exact values of the variables is calculated. Assume that there exists a mathematical expression $x^2 + 2xy + y^2$, and it can be defined as follows.

```
Python console for SymPy 1.0 (Python 2.7.5) These commands
were executed:
from __future__ import division
from SymPy import *
x, y, z, t = symbols('x y z t')
k, m, n = symbols('k m n', integer=True)
f, g, h = symbols('f g h', cls=Function)
x,y=symbols('x y')
x*x+2*x*y+y**2
```

To calculate the exact value of the expression for x=1 and y=2, *subs()* method must be used.

```
x,y=symbols('x y') x*x+2*x*y+y**2
```

$$\boldsymbol{x^2 + 2xy + y^2}$$

```
expr=x*x+2*x*y+y**2
res=expr. subs({x:1,y:2})
res
```

$$\mathbf{9}$$

Firstly, a new label is built to refer to the expression, and then the values are called to variables of the expressions by using *subs()* method. The argument for the *subs()* method is a Python dictionary, which contains two keys and two values. In the last example, a numerical value is substituted for every variable in the expression. In SymPy, any given symbolic value can be substituted for any other symbolic value.

```
expr. subs({x:1-y})
```

$$\boldsymbol{y^2 + 2y(-y + 1) + (-y + 1)^2}$$

If we want to get the solution simplified, for example, when the final solution is a bit complex and when there are some terms which cancel each other out; we may use SymPy’s *simplify()* function, as follows.

```
from SymPy import simplify
simplify(expr. subs({x:1-y}))
                              1
```

The *simplify()* function can also simplfy other complicated expressions such as trigonometric and algorithmic but in this thesis we will not get into these. Now let’s calculate the exact value of a series by using *subs()* function.

```
'''Print the series:
x+x**2/+x**3/3+x**4/4+...+x**n
'''
from SymPy import *
from SymPy import init_printing, Symbol, pprint

def print_series(n, x_value):
  #Initialize printing system in reverse order
  init_printing(order='rev-lex')
  x=Symbol('x')
  series=x
  for i in range(2,n+1):
    series=series+(x**i)/i
  pprint(series)

  #Now Let’s evaluate the series at x value
  series_value=series. subs({x:x_value})
  print('Value of the series at {0}: {1}'.format(x_value,
series_value))


n=input('Enter the number of the terms you want in the series:
')
x_value=input('Enter the value of x: ')
print_series(int(n), float(x_value))
```

Now, the function *print_series()* will have an extra argument that is the value of x, namely x_value. This value will be entered by the user. Second additional line is *series_value=series.subs({x:x_value})* . By using the function *subs()* the exact value can be assigned to a variable.

```
Enter the number of the terms you want in the series:  7
Enter the value of x:  1.2
     2    3    4    5    6    7
    x    x    x    x    x    x
x + ── + ── + ── + ── + ── + ──
    2    3    4    5    6    7
Value of the series at 1.2: 4.52161097142857
```

In this sample run, the code asked for the result from seven terms in the series with 1.2 as x value. So, the program prints the series and calculates the value of the series.

3.1.6 Converting Strings to Mathematical Expressions

Many times mathematical programs take arguments as float or integer from the user. But some other times, it is needed to write more general programs that could handle any given expression given by the user. For this reason, we need to find a way that converts strings to mathematical expressions. SymPy's *sympify()* function can perform this for us. The function is so useful because it converts the strings into SymPy objects which can be used as a mathematical input in a function. Now let's follow the given code.

```
from SymPy import *
from SymPy import sympify, pprint, init_printing

expr=input('Enter the mathematical expression: ')

expr=sympify(expr)
init_printing(order='rev-lex')
```

First of all, import the function *sympify()*. Then take the expression as a string value. Then by using *sympify()* function, the expression is converted from string to symbolic mathematical expression. By using this expression it can be performed various operations. For example, multiply the expression by 2.

```
init_printing(order='rev-lex')
pprint(2*expr)
```

```
Enter the mathematical expression:  x**2+3**y+2*x+x**3
                        2      3      y
                     4·x + 2·x  + 2·x  + 2·3
```

But sometimes user inputs could be invalid expressions. In this kind of situations we can use the *try-except* method, which can be used in any Python code for error handling in user interactions. Let's follow the codes.

```
Enter the mathematical expression:  x**2+2x+x**3
Traceback (most recent call last):
  File "python", line 4, in <module>
SymPy.core.sympify.SympifyError: Sympify of expression 'could
not parse 'x**2+2x+x**3'' failed, because of exception being
raised:
SyntaxError: invalid syntax (<string>, line 1)
```

In the error code, the error line tells us that *sympfy()* can not convert the expression to a mathematical expression because user input has 2x expression. SymPy can not convert the expression because there is no definition for 2x. There is no mathematical operator between 2 and x. So the program will not run and will return the error code. But if we use *SympifyError* exception we can print an error code for user.

```
from SymPy.core.sympify import SympifyError
from SymPy import sympify, pprint
expr=input('Enter the mathematical expression: ')

try:
  expr=sympify(expr)
except SympifyError:
  print('Invalid input')
```

```
Enter the mathematical expression:  x**2+2x+x**3
                        Invalid input
```

Now let's apply the *sympify()* function to write a program which calculates the multiplication of 2 given mathematical expressions.

```
'''
Product of two mathematical expression.
'''
from SymPy import expand, sympify, pprint
from SymPy.core.sympify import SympifyError

def product(equ1, equ2):
  prod=expand(equ1*equ2)
  return prod

equ1=input('Enter the first expression: ')
equ2=input('Enter the second expression: ')

try:
  equ1=sympify(equ1)
  equ2=sympify(equ2)
except SympifyError:
  print('Invalid expression input.')
else:
  pprint(product(equ1, equ2))
```

The last line of the code displays the product of the two expressions. The mathematical inputs don’t have to be in one variable expressions.

```
Enter the first expression:  x**5+3*x**3-7*x**2+15*x+9
Enter the second expression:  x**4+3*x**3
 9      8      7      6      5       4       3
x  + 3·x  + 3·x  + 2·x  - 6·x  + 54·x  + 27·x
```

More than two variables expression example will be

```
Python 3.5.2 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux

Enter the first expression:  x**y+2*x+z
Enter the second expression:  z**x+2*y+x
   2       y                         x      y       y  x
x
2·x  + x·x  + 4·x·y + x·z + 2·x·z  + 2·x ·y + x ·z  + 2·y·z + z·z
```

3.1.7 Equation Solving

SymPy’s *solve()* function is used to solve equations in one variable. When an expression input with a symbol for example with ‘x’, *solve()* function can calculate the value of the variable which makes the equation zero. Writing an equation with an equal sign and the value of zero is not a necessity. Because SymPy automaticaly assume that the function is an equation and it will be solved with respect to the value of the variable. Let’s begin with a simple example.

```
from SymPy import Symbol, solve
x=Symbol(‘x’)
expr=x-5-7
solve(expr)
                            [12]
```

It is clearly seen that the value of the solution is an element of the list. So, the *solve()* function returned a list. Solve function always returns a list because when solving an equation, a solution set is found and there is a rule stating that a solution should always be a natural or rational number. For example, if we try to solve a second degree equation solve function would return a list with two elements. Besides finding all solutions as elements of a list, the solutions could get as a dictionary. Solving a second degree equation with one variable is the following.

```
from SymPy import solve
x=Symbol(‘x’)
expr=x**2+5*x+4
print(solve(expr, dict=True))
                      [{x: -4}, {x: -1}]
```

Firstly, the *solve()* function is imported to the interpreter. Then the variable x is defined and a second degree one variable equation is given as a mathematical expression. The second argument in the solve function is dict. The *“dict=True”* is given because the aim is to get solutions within an order. If there is no solution for the given equation SymPy returnsa an empty list. The roots of the proceeding equation is -4 and -1. Now let’s try this function for another equation.

```
from SymPy import solve
x=Symbol('x')
expr=x**2+x+1
```

```
print(solve(expr, dict=True))
[{x: -1/2 - sqrt(3)*I/2}, {x: -1/2 + sqrt(3)*I/2}]
```

As expected, both roots are imaginary, and the imaginary parts of the solutions is given with the symbol I. In addition, SymPy can manipulate ordinary differential equations, recurrence relations, Diophantine equations and many type of algebraic equations. So far, only the solve function has been used. But SymPy also has another function *solveset()*. There is a very significant difference between the *solve()* and the *solveset()* functions. While the former always returns a list or a dictionary but the latter returns a SymPy set object. But both functions assume that the given function is equal to 0.

Let’s give an example.

```
from SymPy import solve, pprint, solveset
x=Symbol('x')
expr=x**2-2*x*y+1
pprint(solve(expr,x, domain=S.Complexes))
```

$$\left(\boldsymbol{y-\sqrt{y^2-1}, y+\sqrt{y^2-1}}\right)$$

In addition, the roots of the any given second degree equation can be omitted with respect to the coefficients of the equation. Now take a second degree equation as $ax^2 + bx + c = 0$ and try to find all the roots of the equation with respect to a,b,c.

```
from SymPy import solve, pprint, solveset
x,a,b,c=symbols('x a b c')
expr=a*x**2+b*x+c
print(solve(expr,x, dict=True))
[{x: (-b + sqrt(-4*a*c + b**2))/(2*a)}, {x: -(b + sqrt(-4*a*c
+ b**2))/(2*a)}]
```

Now, consider an example from Physics. The motion equation will be used. According to equation of motion, the distance travelled by a body can be calculated by using the constant acceleration ‘a’, initial velocity ‘u’ and time ‘t’. If the equation is organized, $s = ut + \frac{1}{2}at^2$ is founded. An example code snippet will look like the following.

```
from SymPy import solve, pprint, solveset
s,u,t,a=symbols('s u t a')
expr=u*t+(1/2)*a*t*t-s
t_expr=solve(expr,t, dict=True)
pprint(t_expr)
```

The solution set will be

```
⎡⎧                                ⎫  ⎧     ⎛                                ⎞ ⎫⎤
⎢⎪              ________________ ⎪  ⎪     ⎜           ________________     ⎟ ⎪⎥
⎢⎪             ╱               2 ⎪  ⎪     ⎜          ╱               2     ⎟ ⎪⎥
⎢⎨     -u + ╲╱   2.0⋅a⋅s + u    ⎬  ⎨   - ⎝u + ╲╱   2.0⋅a⋅s + u    ⎠ ⎬⎥
⎢⎪t: ─────────────────────────── ⎪, ⎪t: ─────────────────────────────────── ⎪⎥
⎣⎩               a               ⎭  ⎩                  a                   ⎭⎦
```

Finding the solution set of a system of a linear equation is also possible in SymPy. Now, let’s show this property with an example.

```
from SymPy import solve, pprint, solveset
x,y=symbols('x y')
expr1=2*x+3*y-11
expr2=3*x-12*y+6
pprint(solve((expr1, expr2),dict=True))
```

Then the solution will be

```
[{x: 38/11, y: 15/11}]
```

In the given equation systems, the solutions are also checked. Consider the previous system of equations.

```
from SymPy import solve, pprint, solveset
x,y=symbols('x y')
expr1=2*x+3*y-11
expr2=3*x-12*y+6
soln=solve((expr1, expr2),dict=True)
soln=soln[0]
chck1=expr1.subs({x:soln[x],y:soln[y]})
print(chck1)
                                            0
chck2=expr2.subs({x:soln[x],y:soln[y]})
print(chck2)
                                            0
```

The both results will give zero as expected.

3.1.8 Plotting by SymPy

By using SymPy, the graph of any given equation can also be drawn. In Anaconda and SymPy, you don't have to import anything but in any other IDE an import statement must be added for the *mathplotlib library*. And then, we also don't have to add *show()* function to show the graph because this could be automatically done by SymPy. Consider the following example.

```
from SymPy import *
from SymPy.plotting import plot
x,y=symbols('x y')
plot(2*x+3)
```

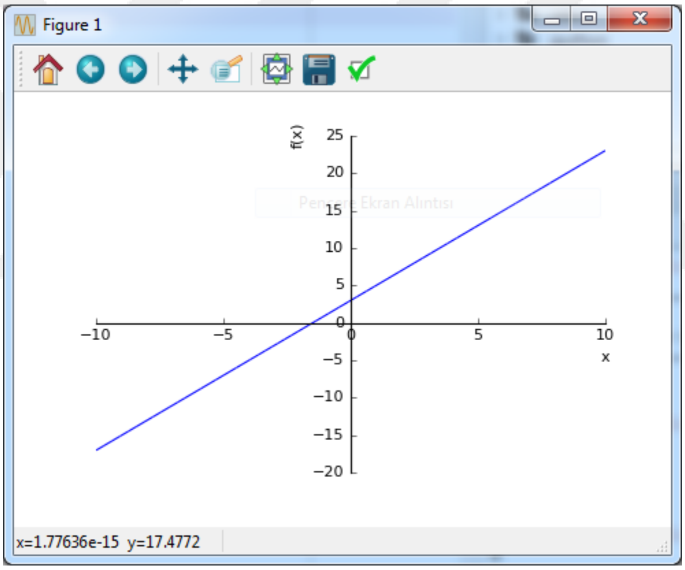


Figure 3.1. Graph *y=2x+3.*

The graph shows that the default range of the x and y is automatically taken as -10 and 10. This values can also be changed as the following code snippet shows.

```
from SymPy import *
from SymPy.plotting import plot
x,y=symbols('x y')
plot((2*x+3), (x,-5,5))
```

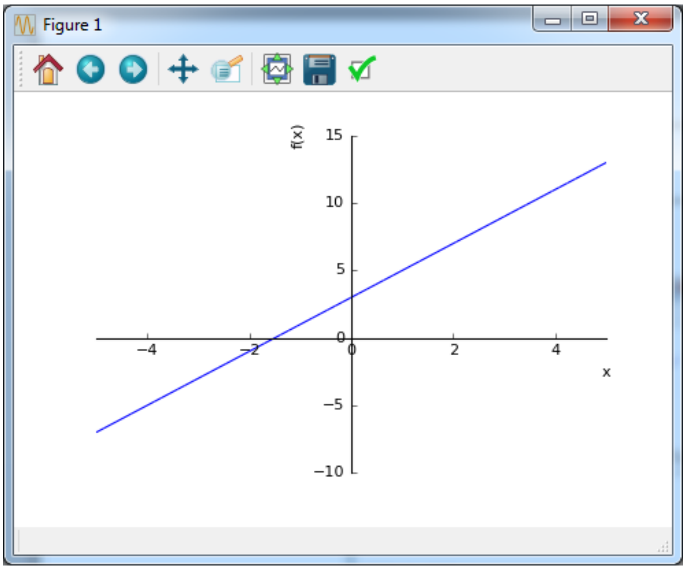


Figure 3.2. Graph *y=2x+3* for *x* in *(-5,5)*.

SymPy has extra opportunities to add many details to a graph. For example by using an extra line and some arguments, labels and a title can be added to a graph.

```
from SymPy import *
from SymPy.plotting import plot
x,y=symbols('x y')
plot((2*x**2+3*x-5), (x,-5,5), title='A Graph', xlabel='x',
ylabel='2*x**2+3*x-5')
```

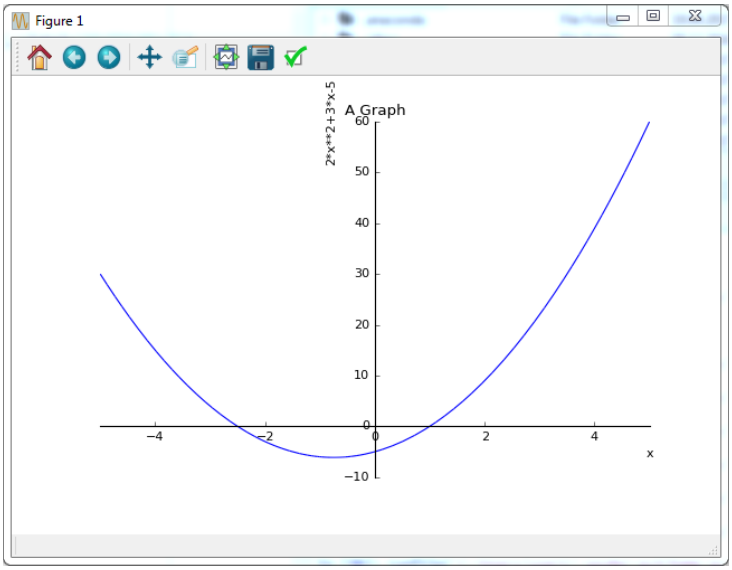


Figure 3.3. Graph of a second degree polynomial.

Also a program which takes mathematical expressions (equations) from the user and plots them can be written.

```
'''
user input graph plotting
'''
from SymPy import *
from SymPy.plotting import plot

def graph_plotter(expr):
    x,y=symbols('x,y')
    solutions=solve(expr, y)
    expr_y=solutions[0]
    plot(expr_y)

expr=input('Enter your equation in terms of x and y: ')
try:
    expr=sympify(expr)
except:
    print('Input is not a mathematical expression.')
else:
    graph_plotter(expr)
```

```
Enter your equation in terms of x and y: x**3+3*x**2+2*x+3-y
```

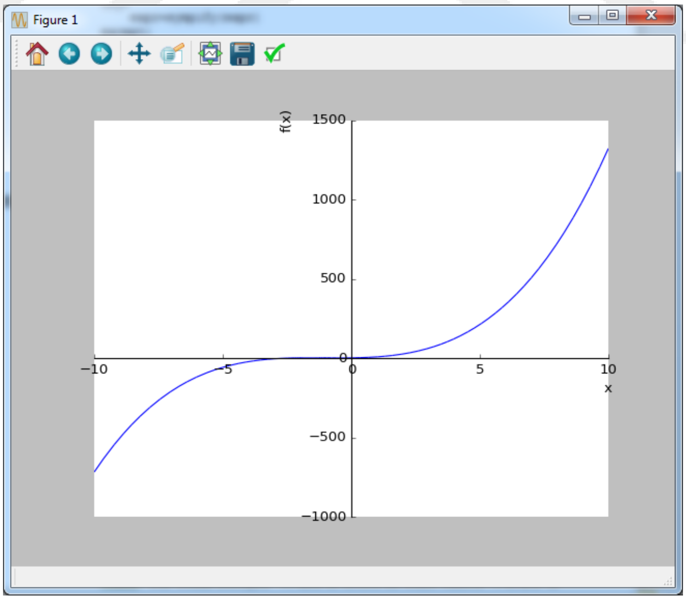


Figure 3.4. Graph of one function.

On the same graph more than one equation can be shown and more than one extra labels and colors can be used.

```
'''
more than one plotting
'''
from SymPy import *
from SymPy.plotting import plot
x,y=symbols('x y')
p=plot(3*x**2+2*x+3, 3+2*x-x**2, legend=True, show=False)
p[0].line_color='blue'
p[1].line_color='red'
p.show()
```

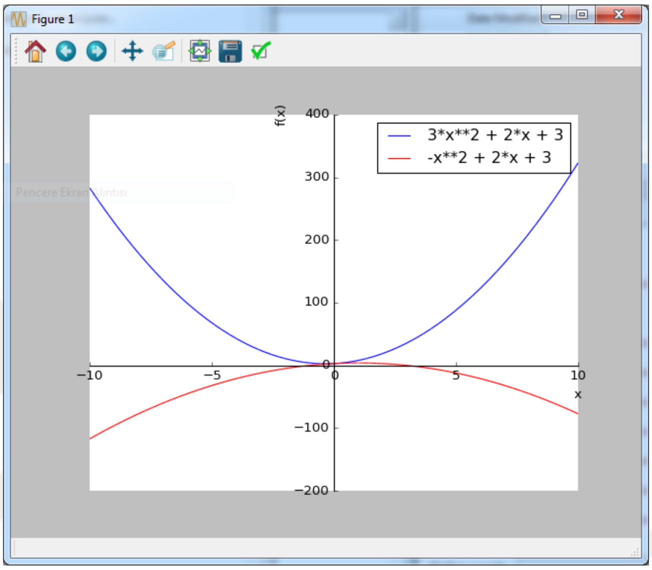


Figure 3.5. Graph of two functions.

In this chapter, the basics of the symbolic mathematics using SymPy have been given, such as declaring the symbols, constructing the mathematical expressions by using these symbols, using mathematical operators, solving equations, linear equation systems and plotting graphs. In the following examples, includes some challenges.

```
# Factor Finder

import SymPy
from SymPy import factor, sympify

def factor_finder(expr):
    nexpr=sympify(expr)
    return factor(nexpr)
expr=input('Enter your expression: ')

print(factor_finder(expr))
```

```
#Graphical Equation Solver
from SymPy import *
from SymPy import sympify, symbols,solve
from SymPy.plotting import plot


expr1=input('Enter first equation in terms of x and y: ')
expr2=input('Enter second equation in terms of x and y: ')

def ges(expr1, expr2):
    x,y=symbols('x y')
    expr1=sympify(expr1)
    expr2=sympify(expr2)
    solution1=solve(expr1,y)
    solution2=solve(expr2,y)
    expr1_y=solution1[0]
    expr2_y=solution2[0]
    inter=expr1_y-expr2_y
    soln=solve(inter,dict=True)
    p=plot(expr1_y, expr2_y,legend=True, show=False)
    p[0].line_color='b'
    p[1].line_color='r'
    print(soln)
    p.show()
try:
    expr1==sympify(expr1) and expr2==sympify(expr2)
except ValueError:
    print('Invalid')
else:
    ges(expr1,expr2)
```

```
# Finding sum of a given arbitrary series
#summation() used instead of a loop

from SymPy import *
from SymPy import init_session

def series_sum(expr,term):
    a,n,d=symbols('a n d')
    expr=sympify(expr)
    s=summation(expr,(n,1,term))
    print(s)


if __name__=='__main__':
    expr=input('Enter your series in terms of a,n,d: ')
```

```
    term=int(input('Enter the number of terms: '))

series_sum(expr, term)
```

```
# Single variable polynomial inequality solver
from SymPy import *
from SymPy import init_session

def PolySolver(expr):
    x=Symbol('x')
    expr=sympify(expr)#sympfying the user input.
    ineq=expr
    lhs=ineq.lhs#Extract the left side.
    p=Poly(lhs,x)#Creating a polynomial object
    rel=ineq.rel_op#Extract the relational operator from the
ineq. obj.
    print(solve_poly_inequality(p,rel))

if __name__=='__main__':
    print('Single Variable Inequality Solver')
    expr=input('Enter inequality: ')
PolySolver(expr)
```

```
# Single variable rational inequality solver
from SymPy import *
from SymPy import init_session

def RatSol(expr):
    x=Symbol('x')
    ineq=sympify(expr)
    lhs=ineq.lhs
    numer, denom=lhs.as_numer_denom()
    p1=Poly(numer)
    p2=Poly(denom)
    rel=ineq.rel_op
    print(solve_rational_inequalities([[((p1,p2), rel)]]))

if __name__=='__main__':
    print('Single variable rational inequality solver.')
    expr=input('Enter your inequalities in form f(x)/g(x): ')
RatSol(expr)
```

In recent four challenges the *try-except* method was not used and *is_polynomial()* function was also not used to check whether the given function is a polynomial or not. Moreover, functions *is_rational_functionl()* can also be used to control but the use of this function was also not preferred.

3.2 Calculus with SymPy

In this section the main objective is to solve calculus problems using SymPy functions. First, the definition of the mathematical mean of the functions will be given. Then the most common used mathematical functions available in standart Python's library and SymPy will be given. Finding the limits of a function, calculating derivatives and calculating integrals will also be given in this chapter. Since the basic concepts and assumptions have already been given in the previous section, it is not considered to be appropriate to repeat them in this section.

3.2.1 Basic Definitions

The definitions of the function, limit, derivative and integral are given below.

Definition: Let A and B be sets. A function from A to B is a relation, f, from A to B such that if for $a \epsilon A$ and $b, c \epsilon B, (a,b) \epsilon f$ and $(a,c) \epsilon f$, then $b = c$. If $(a,b) \epsilon f$, then we write $b = f(a)$. A function from A to B is also called a mapping from A to B.

Definition: If f is a function from A to B then

i. the domain of f, written $Dom(f)$, is the set:$Dom(f) = \{a \epsilon A |\ there\ exists\ b \epsilon B\ such\ that\ b = f(a)\}$.

ii. the range of f, written $Ran(f)$, is the set: $Ran(f) = \{b \epsilon B |\ there\ exists\ a \epsilon A\ such\ that\ b = f(a)\}$.

When considering a function from A to B, it is assumed that A=Dom(f). In all cases $f: A \rightarrow B$ will be used to denote a function.

Definition: Let f(x) be defined in a deleted neighbourhood of the point a. Then

$$\lim_{x \to a} f(x) = L$$

means that given any $\varepsilon > 0$ (no matter how small), we can find a (sufficiently small) $\delta > 0$ such that

$$|f(x) - L| < \varepsilon$$

whenever

$$0 < |x - a| < \delta.$$

Definition: Let f be a function defined in a neighborhood of a point x. Then by the derivative of f at x, denoted by $f'(x)$, it is meaned the limit

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x},$$

Provided that the limit exists, or equivalently

$$f'(x) = \lim_{u \to x} \frac{f(u) - f(x)}{u - x},$$

(let $u = x + \Delta x$). If f has derivative at x, we also say that f is differentiable at x.

3.2.2 Finding Limits

A basic task in calculus is finding the limit values of the function. For a given variable assumed to approach a certain value, the limit of a function can be calculated. Assume that the limit value of the function f(x)=1/x, as x goes to infinity is needed, whose graph is given below.

When the x value is maximized (or approaching the infinity) f(x) approaches the zero. Using the limit notation, it can be written as.

$$\lim_{x\to\infty} \frac{1}{x} = 0$$

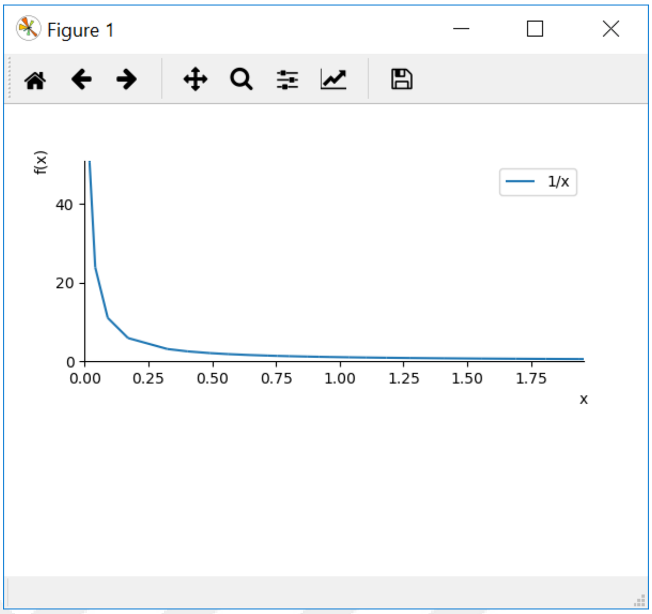


Figure 3.6. Graph of *1/x.*

The limit of the function can be found by using SymPy. Firstly, an object of the *Limit class* is created as follows.

```
from SymPy import Limit, Symbol, S
x=Symbol('x')
print(Limit(1/x, x, S.Infinity))
```

At first line, Limit and Symbol classes imported as well as S, which is a special classes because it contains the definition of positive and negative infinity. The result line will be as follows.

```
Limit(1/x, x, oo, dir='-')
```

As expected from the first three lines, there would not be an evaluated value. The symbol oo denotes positive infinity and the *dir='-'* symbol specifies that x value approaches the point where the limit is searched for from the negative side. So, in order to evaluate the value of the limit, the *doit()* function must be used.

```
from SymPy import Limit, Symbol, S
x=Symbol('x')
L=Limit(1/x, x, S.Infinity)
print(L.doit())
```

```
0
```

As default, the limit value is found from positive direction. But, the default direction can be changed as folllows.

```
from SymPy import Limit, Symbol, S
x=Symbol('x')
L=Limit(1/x, x, 0, dir='-')
print(L.doit())
                             -oo
```

Here the value of

$$\lim_{x\to 0}\frac{1}{x}$$

is calculated and as *x* approaches to 0 from the negative side, the value of the limit approaches negative infinity. On the other side, if *x* approaches to 0 from the positive side, the value will approach the positive infinity.

```
from SymPy import Limit, Symbol, S
x=Symbol('x')
L=Limit(1/x, x, 0, dir='+')
print(L.doit())
                              oo
```

The limit class can also handles the indeterminate forms of the function as

$$\frac{0}{0}\ and\ \frac{\infty}{\infty}.$$

Let’s take the function while x approaches the zero and the value of the function at zero equal to *0/0.*

```
from SymPy import Limit, Symbol, S
from SymPy import sin
L=Limit(sin(x)/x, x,0)
print(L.doit())
                              1
```

Generally, the L’Hospital Rule is used for solving this type of undefined limits but as expected SymPy automaticaly evalutes the value of the limit because the *Limit* class takes care of this for us.

3.2.3 Continuous Compound Calculation

The genius mathematician James Bernoulli found that while the value of n is increasing, the term $(1 + 1/n)^n$ approaches the value of *e* such that the constant can be verified by finding the limit of the given function using SymPy.

$$S = \left(1 + \frac{1}{n}\right)^n$$

```
from SymPy import *
from SymPy import init_session
n=Symbol('n')
L=Limit((1+1/n)**n, n, S.Infinity).doit()
print(L)
```

E

By using this function the continuous compound of interest can be calculated. Let’s assume that the principal amount of money p, rate r, and any number of years t, the interest can be calculated by the formula as follows.

$$S = P \cdot \left(1 + \frac{r}{n}\right)^{nt}.$$

If the S function converted to python code it will be as follows.

```
from SymPy import symbols, Limit, S
n,p,r,t=symbols('n p r t', positive=True)
L=Limit(p*(1+r/n)**(n*t), n, S.Infinity).doit()
print(L)
```

$\mathbf{p * e^{r*t}}$

First, three symbols objects and *n* are created. Then the sign of these symbol objects are defined in *symbols()* function as *positive=True*. If *positive=True* is not written, SymPy would not know anything about the numerical values of the symbol which is assumed and would not be able to calculate the limit value of the given expression.

So far, the value of the limit for any given mathematical expression is calculated while a variable of the function approaching the exact value of a real number or infinity. Now, the next step is the finding the derivative of the function. Now, try to find the derivative of any given function by using the definition.

Consider an object moving in the road. The function of the distance with respect to time is given as

$$S(t) = 3t^2 + 5t + 8.$$

In this function the independent variable is *time-t* because it represents the elapsed time since the object has started to move. If we measure the instantenous rate of change of distance between t_1 and t_2, a new expression will be as follows

$$\frac{S(t_2) - S(t_1)}{t_2 - t_1}.$$

This is also referred as an avarage rate of change of the function. Let's assume that the time distance between t_2 and $t_1$is so small as δ. So, the last expression can be changed as

$$\frac{S(t_1 + \delta_t) - S(t_1)}{\delta_t}.$$

This expression is also a function with t_1 as the variable. If the value of the δ_t is very small, such that it approaches to zero, the limit notation can be used to write

$$\lim_{\delta_t \to 0} \frac{S(t_1 + \delta_t) - S(t_1)}{\delta_t}.$$

Now, evaluate the last limit expression.

```
from SymPy import Symbol, Limit, S, pprint
t=Symbol('t')
St=3*t**2+5*t+8
t1=Symbol('t1')
delta_t=Symbol('delta_t')
St1=St.subs({t:t1})
St1_delta=St.subs({t:t1+delta_t})
L=Limit((St1_delta-St1)/delta_t, delta_t,0).doit()
print(L)
```

$$\boldsymbol{6t_1 + 5}$$

The limit calculated in the last codes snippet is referred as the derivative of the function and it is written by using the definition of the derivative. In SymPy, we don’t have to write these codes always because the *Derivative* class can calculate the derivative easily.

3.2.4 Calculating the Derivative

In SymPy the *Derivative class* can handle the derivation. But an instance of the derivative class has to be created to find the derivative of any given function. Now, consider the previous example representing the motion and time function of an object.

```
from SymPy import Symbol, Derivative
t=Symbol('t')
St=3*t**2+5*t+8
D=Derivative(St,t)
print(D.doit())
```

The result will be *6t+5* as expected. The derivative at given any point by using *subs()* function can also be calculated.

```
from SymPy import symbols, Derivative
t, t1=symbols('t , t1')
St=3*t**2+5*t+8
D=Derivative(St,t)
print(D.doit().subs({t:t1}))
                           6*t1 + 5
print(D.doit().subs({t:1.2}))
                       12.2000000000000
```

Now let’s try for a complicated function whose only variable is x.

```
from SymPy import symbols, Derivative
x, y=symbols('x , y')
Sx=(x**5-3*x**2-7*x)*(x**4-2*x-x)
D=Derivative(Sx,x)
print(D.doit())
(4*x**3 - 3)*(x**5 - 3*x**2 - 7*x) + (x**4 - 3*x)*(5*x**4 -
6*x - 7)
```

As seen in the last example SymPy can handle the derivative of a product of two or more functions. The derivatives of more complicated functions which involves trigonometric functions could also be founded. The codes can be extended such that one can input the function. Let’s write a derivative calculator program. But there will be a little trick because this program asks for the variable name from the user.

```
from SymPy import Symbol, Derivative, pprint, sympify
from SymPy.core.sympify import SympifyError

def derivative(f,var):
    var=Symbol(var) #not var=Symbol(‘var’)
    D=Derivative(f,var).doit()
    pprint(D)

if __name__=='__main__':
    f=input('Enter a function: ')
    var=input('Enter the variable: ')
    try:
        f=sympify(f)
    except SympifyError:
        print('Invalid Input')
    else:
        derivative(f, var)
```

```
Enter a function: 3*x**3+2*x

Enter the variable: x
```

$$9 \cdot x^2 + 2$$

At this point an important coding rule will be given. When you write on the IP(Interactive Python) IDLE like Spyder which is an official scientific Python idle of Anaconda, x=Symbol(‘x’) it is considered to be valid. But, when you try this on the core (not Shell or IP) you have to write x=Symbol(x) in the code lines. Otherwise, program will calculate the derivative of the function with respect to x as 0.

Let’s see a sample run for this common mistake.

```
Enter a function: x**3+2*x+1

Enter the variable: x
                              0
```

3.2.5 Partial Derivative Calculation

In the previous example, it is aimed to find the derivative of a given function with only one variable x. But functions may contain more than one variable and the derivative of the function could be try to find due to an existing variable. This calculation is generally called as *partial differentiation*, with partial indicating.

Let’s assume that the function $f(x, y) = x^3y^2 + 2x$. The partial differentiation of $f(x, y)$ wrt x is:

$$\frac{\partial f}{\partial x} = 3x^2y^2 + 2.$$

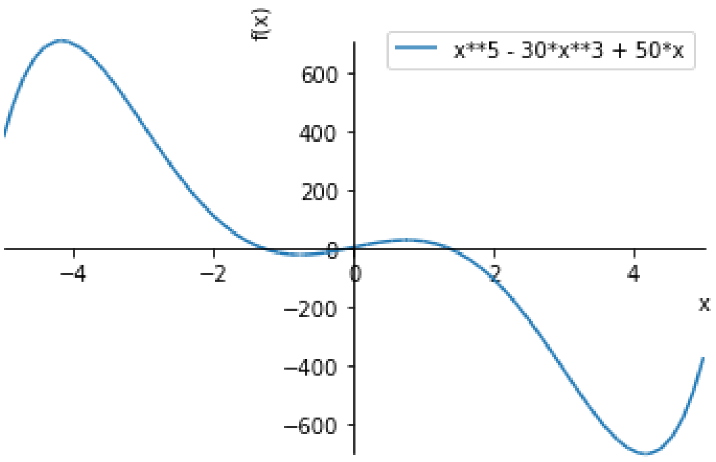
Our last example is capable to find the partial derivative because the *Derivative()* functions consist of an element *var*. Let’s give an example.

```
Enter a function: 3*x**3*y**2+2*y

Enter the variable: y
```

$$\mathbf{6 \cdot x^3 \cdot y + 2}$$

3.2.6 Calculating Higher Order Derivatives and Maxima-Minima

```
from SymPy import Symbol
from SymPy. plotting import plot
x=Symbol('x')
p=plot(x**5-30*x**3+50*x,(x,-5,5), legend=True, show=False)
print(p.show())
```



In the above example, there exists a function and its graph for $-5 \leq x \leq 5$. There are some bending points on the graph. These points are called as the maxima, minima, local maxima, and local minima points. As seen on the graph at that points the derivative will be equal to 0. Because of the definition, it is also said that the derivative is zero. By default, *Derivative* class finds the first-order derivative. To find higher order derivatives, SymPy gives an option in *Derivative* class as the third argument. In this section, the higher order derivatives and extremum points will be found.

The following example find the critical points of a given function.

```
from SymPy import Symbol, solve, Derivative
x=Symbol('x')
f=x**5-30*x**3+50*x
d1=Derivative(f,x).doit()
print(d1)
critical_points=solve(d1)
print(critical_points)
                    5*x**4 - 90*x**2 + 50
[-sqrt(-sqrt(71) + 9), sqrt(-sqrt(71) + 9), -sqrt(sqrt(71) +
9), sqrt(sqrt(71) + 9)]
```

The critical points which are found here, are assign to the letters B, C, A and D. Let’s create labels to refer to these points.

```
from SymPy import Symbol, solve, Derivative
x=Symbol('x')
f=x**5-30*x**3+50*x
d1=Derivative(f,x).doit()
print(d1)
critical_points=solve(d1)
A=critical_points[2]
B=critical_points[0]
C=critical_points[1]
D=critical_points[3]
```

All of the critical points lie between the points 5 and -5. To find the global maximum and global minimum of f(x), the second derivative test should be used. By using this test, the critical points which are maxima or minima can be determined. First, calculate the second derivative of the function.

```
from SymPy import Symbol, solve, Derivative
x=Symbol('x')
f=x**5-30*x**3+50*x
x=Symbol('x')
p=plot(x**5-30*x**3+50*x,(x,-5,5), legend=True, show=False)
p.show()

d1=Derivative(f,x).doit()
print(d1)
p1=plot(d1,(x,-5,5), legend=True, title=('Derivative of f'),
show=False)
p1.show()
critical_points=solve(d1)
A=critical_points[2]
B=critical_points[0]
C=critical_points[1]
D=critical_points[3]
d2=Derivative(f,x,2).doit()
for point in critical_points:
    if d2.subs({x:point}).evalf()<0:
        print("{} is local maximum".format(point))
    elif d2.subs({x:point}).evalf()>0:
        print("{} is local minimum".format(point))
    else:
        print("{} is inconclusive".format(point))
                    5*x**4 - 90*x**2 + 50
-sqrt(-sqrt(71) + 9) is local minimum
sqrt(-sqrt(71) + 9) is local maximum
-sqrt(sqrt(71) + 9) is local maximum
sqrt(sqrt(71) + 9) is local minimum
```

For the function $f(x) = e^x$, there might not be any critical points in the domain, but in this case the method works fine: indeed, it says us the extrema occur at the domain boundary.

3.2.7 Integral

The indefinite integral, or the antiderivative, of a function f(x) is another function F(x), such that $F'(x) = f(x)$. Mathematically it is written as

$$F(x) = \int f(x)dx.$$

The definite integral, on the other side is the integral

$$\int_{a}^{b} f(x)dx,$$

which is equal to $F(b) - F(a)$, where $F(b)$ and $F(a)$ are the values of the antiderivative at the points a and b. If one want to calculate this *definite integral*, she has to create *Integral* object for both value.

Now, let’s begin with a simple integral which is $\int kxdx$, where k is an arbitrary constant.

```
from SymPy import symbols, Integral, pprint
x,k=symbols('x k')
I=Integral(k*x,x)
pprint(I)
```

$$\int \boldsymbol{kxdx}$$

As seen on the code block, the codes do not generate a solution because we just have been written only the integral. We did not want the solution. For this reason we have to add *doit()* function to code.

```
from SymPy import symbols, Integral, pprint
x,k=symbols('x k')
I=Integral(k*x,x)
pprint(I.doit())
```

$$kx^2/2$$

If it is aimed to get the solution as a definite integral, the upper and lower bounds of the integral must be added.

```
from SymPy import symbols, Integral, pprint
x,k=symbols('x k')
I=Integral(k*x,(x,2,6))
pprint(I.doit())
```

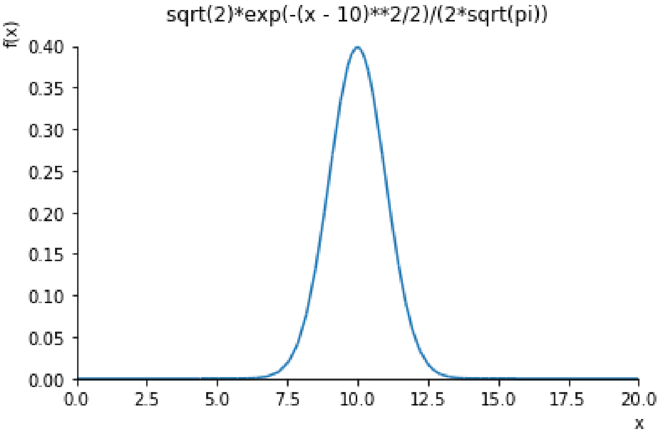
$$16 \cdot k$$

The probability density function, f(x), express the probability of the value of a random variable being close to x, an arbitrary value. It can also tell us the probability of x falling within an interval. The probability density function defined as

$$\frac{1}{\sqrt{2\pi}}e^{-\frac{(x-10)^2}{2}}$$

The given graph below is the graph of the function f.

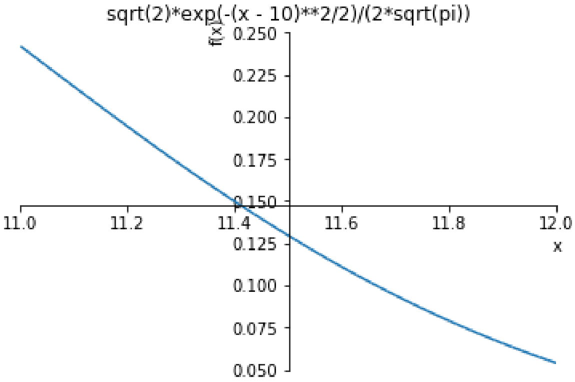
```
#Probability Density Function
from SymPy import *
D=exp(-(x-10)**2/2)/sqrt(2*pi)
p=plot(D, (x,0,20), legend=False, show=False, title=D)
p.show()
```



If you want to calculate

$$f(11 < x < 12) = \frac{n(E)}{n(S)},$$

```
from SymPy import *
D=exp(-(x-10)**2/2)/sqrt(2*pi)
p=plot(D, (x,11,12), legend=False, show=False, title=D)
p.show()
```



```
I=Integral(D, (x,11,12)).doit().evalf()
print(I)
```

```
0.135905121983278
```

Thus the probability which could be a grade of a coding lecture lies between 11 and 12 is so close to 0.14. The function is evaluated by *doit()* function and found the numerical value using *evalf()*.

A probability density function has two basic properties: the first one is the value of the x which is greater than zero. It can not be smaller than zero. And the value of the definite integral

$$\int_{-\infty}^{\infty} f(x)dx = 1.$$

If we calculate the value of this integral,

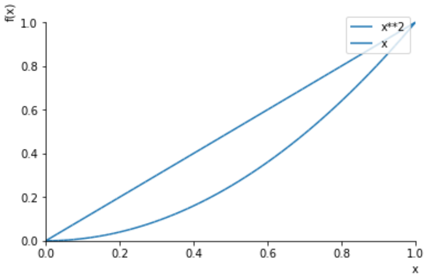
```
# -*- coding: utf-8 -*-
"""
Created on Tue May  2 23:17:27 2017
@author: fatih.cansu
"""
from SymPy import *
x=Symbol('x')
p=exp(-(x-10)**2/2)/sqrt(2*pi)
I=Integral(p,(x,S.NegativeInfinity,S.Infinity)).doit().evalf()
print(I)
                                1.00000000000000
```

In this section, we have been doing limits, derivatives, and integrals of functions by coding. Now let us assume that two functions are given by the user input and our aim is finding the area between two curves. It is clear that the area between the curves f(x) and g(x) is

$$\int_{a}^{b} \big(f(x) - g(x)\big)dx.$$

The points a and b are the intersection points such that $a < b$. The function f is the upper function and the g is the lower function. Our challenge is the code a program that will allow the user to input any two single variable functions. The critical point in this program is making it clear that the first function entered should have a greater value, and should ask for the values of x.

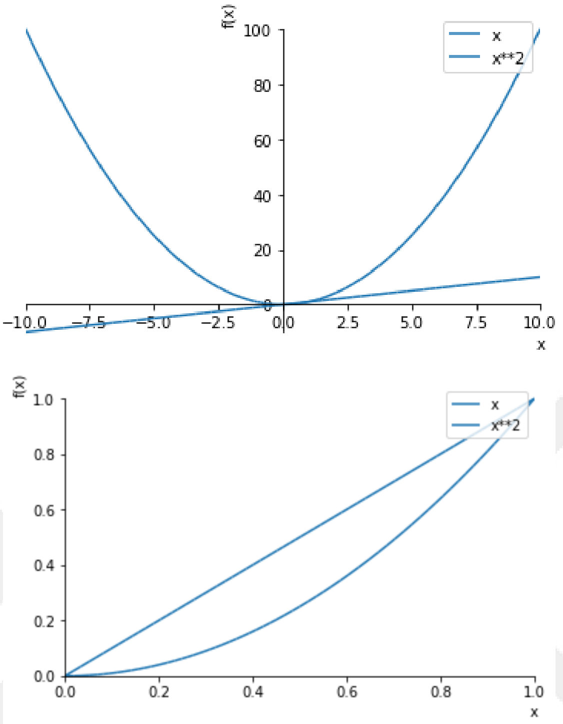
```
# -*- coding: utf-8 -*-
"""Created on Tue May  2 23:17:27 2017
@author: fatih.cansu"""
from SymPy import symbols, Integral, pprint,plot, solve
x,y=symbols('x y')
f=x**2 #Example f
g=x #Example g
h=f-g
solutionset=solve(h,x)
down=solutionset[0]
up=solutionset[1]
p=plot(f,g,(x,down,up), legend=True, show=False)
p.show()
I=Integral(h,(x,up,down))
pprint(I.doit())
```



```
1/6
```

```
# -*- coding: utf-8 -*-
"""Created on Tue May  2 23:17:27 2017
@author: fatih.cansu
Area betwen two curve
"""
#User defined f and g
from SymPy import symbols, Integral,plot, solve,sympify,
SympifyError
x,y=symbols('x y')
def area_between_curves(f,g):
    h=f-g
    solutionset=solve(f-g,x)
    down=solutionset[0]
    up=solutionset[1]
    p=plot(f,g, legend=True, show=False)
    p.show()
    p=plot(f,g,(x,down,up), legend=True, show=False)
    p.show()
    I=Integral(h,(x,up,down)).evalf()
    return abs(I.doit())

if __name__=='__main__':
    f=sympify(input("Enter your first curve: "))
    g=sympify(input("Enter your second curve: "))
```

```
    try:
        (f==sympify(f) and g==sympify(g))
    except SympifyError:
        print('Invalid Input')
    else:
        print(area_between_curves(f,g))
```



```
1/6
```

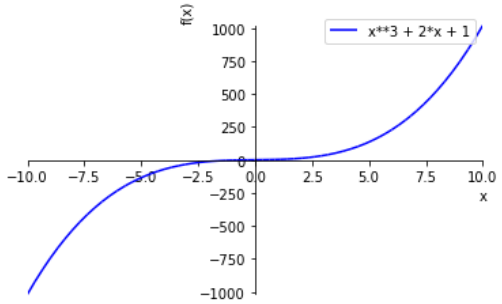
Now let us calculate the length of the arc between any given two points for an arbitrary function, *f(x)*.

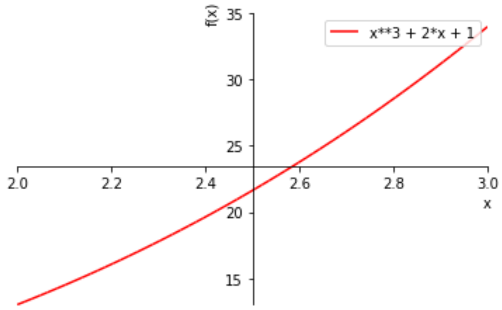
```
# -*- coding: utf-8 -*-
"""
Created on Thu May  4 11:22:53 2017

@author: fatih.cansu
Find the length of a curve between two points
"""
from SymPy import *

def curve_length(f,var,a,b):
    var=Symbol(var)
    p=plot(f,legend=True, show=False)
    p[0].line_color='blue'
    p.show()
    p=plot(f,(var,a,b), legend=True, show=False)
    p[0].line_color='red'
    p.show()
    D=Derivative(f,var).doit()
```

```
    Len=Integral(sqrt(1+D**2), (var, a, b)).doit().evalf()
    return str(Len)[0:7]

if __name__=='__main__':
    f=input("Enter your curve(in one variable): ")
    var=input("Enter the variable: ")
    a=float(input('Enter down bound: '))
    b=float(input('Enter upper bound: '))
    print("The length of {0} between {1} and {2} is:
{3}".format(f,a,b,curve_length(f,var,a,b)))
```

Let run the code for a sample function.

```
Enter your curve(in one variable): x**3+2*x+1

Enter the variable: x

Enter down bound: 2

Enter upper bound: 3
```





```
The length of x**3+2*x+1 between 2.0 and 3.0 is: 21.0248
```

As a last example, an interesting shape and its volume and, surface area will be given. *The Gabriel's Horn* is a kind of geometrical shape with interesting and paradoxial properties. Its surface area is inifine but it has finite volume. First let us give the mathematical proofs.

Let us consider that surface area and the volume of the solid built by rotating the line y=1/x around x-axis. The bound of the rotation is x-axis and x=1 line. The volume of that solid by revoluation can be calculated by using shell method. So

$$V = \pi \int_{1}^{a} \frac{dx}{x^2} = \pi \left(1 - \frac{1}{a}\right).$$

If it is assumed as *a* approaches to the *infinity*,

$$\lim_{a \to \infty} \pi \left(1 - \frac{1}{a}\right) = \pi.$$

It will be found as expected the volume of the horn to be finite and equal to *pi*. Now, let's look for the value of surface area. The surface area of any given solid is

$$S = 2\pi \int_{a}^{b} r(x) \sqrt{1 + \left(\frac{dy}{dx}\right)^2}\, dx.$$

Since the value of $\frac{d}{dx}\frac{1}{x} = -\frac{1}{x^2}$, the surface area formula will be

$$S = 2\pi \int_{1}^{a} \frac{1}{x} \sqrt{1 + \left(-\frac{1}{x^2}\right)^2}\, dx$$

$$= 2\pi \int_{1}^{a} \frac{1}{x} \sqrt{1 + \frac{1}{x^4}} dx.$$

Instead of calculating integral value the inequalities method can be used to show the surface area is unbounded. Since the interval is *(1,a),* the expression in the square root and the *r(x)* are positive.

$$2\pi\int_1^a \frac{1}{x}\sqrt{1+\frac{1}{x^4}}\,dx \geq 2\pi\int_1^a \frac{1}{x}\,dx.$$

From this inequality, it can be written

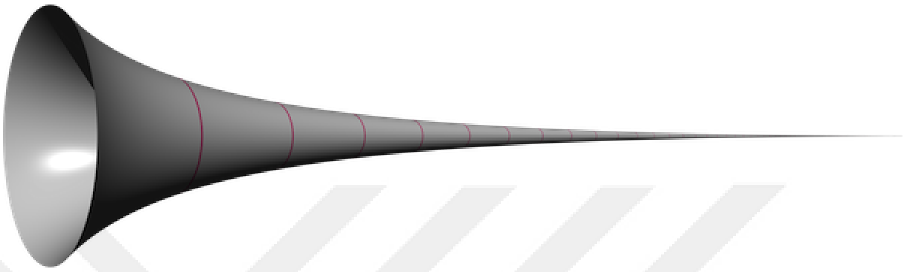


Figure 3.7. Gabriel's Horn.

$$S \geq 2\pi\int_1^a \frac{1}{x}\,dx = 2\pi ln a.$$

If the limit value of the right hand side with approaches to infinity is calculated, an impossible inequality $S \geq \infty$ will be found. Now, let us check this paradoxials in SymPy.

```
# -*- coding: utf-8 -*-
"""
Created on Thu May  4 17:23:11 2017

@author: fatih.cansu
Volume of the Gabriel's Horn
"""
import SymPy
from SymPy import *
x,y,a=symbols('x y a')
f=1/x
I=pi*Integral(f**2,(x,1,a)).doit()
pprint(I)
L=Limit(I,a,S.Infinity).doit()
pprint(L)
```

```
π(1 − 1/a)
π
```

```
# -*- coding: utf-8 -*-
"""
Created on Thu May  4 17:23:11 2017

@author: fatih.cansu
Area of the Gabriel's Horn
"""
import SymPy
from SymPy import pi, Derivative, symbols, Integral, S, sqrt,
Limit, pprint
x,y,a=symbols('x y a')
f=1/x
I=pi*Integral(f**2,(x,1,a)).doit()
L=Limit(I,a,S.Infinity).doit()
S=2*pi*Integral(f*sqrt(1+Derivative(f,x)**2),(x,1,S.Infinity))
.doit()
pprint(S)
```

```
                    Integral does not convergent
```

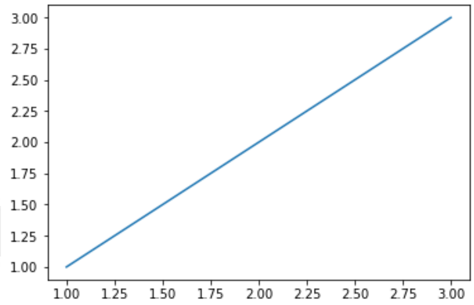
3.3 Fractals and Geometric Shapes

In this section it will be discussed how the basic geometric shapes are drawn like circles, triangles, and the other polygons. In the last part of the section, fractals will be constructed by using codes, the complex geometric shapes like fractals will be constructed by very basic and simple but clever algorithms and, repeated applications of simple geometric transformations.

3.3.1 Geometric Shapes with Mathplotlib and Patches

In *SymPy* drawing the graph of any given equations is already discussed in previous sections. Now it will be tried to draw graphs and geometric shapes without SymPy libraries. Instead of *SymPy*, mathplotlib and its patches will be used. In *Mathplotlib*, the patches allow us to draw geometric shapes. First let us try to understand how a *matplotlib* plot is constructed. Assume that there exists a line passing *(1,1), (2,2), (3,3),* and *(4,4).*

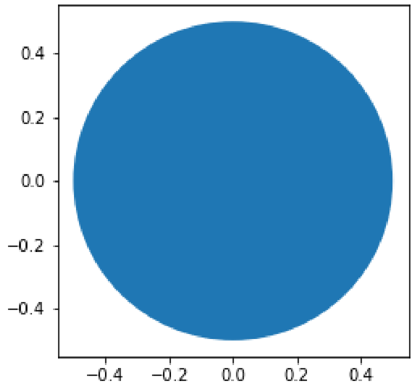
```
# -*- coding: utf-8 -*-
"""
Created on Fri May  5 14:28:37 2017

@author: fatih.cansu
"""
import matplotlib.pyplot as plt
x=[1,2,3]
y=[1,2,3]
plt.plot(x,y)
plt.show()
```



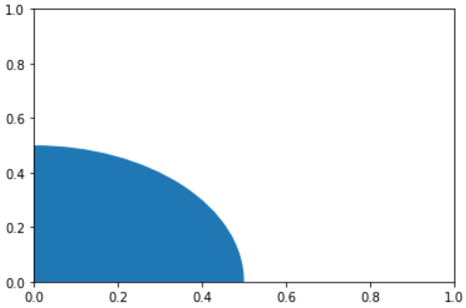
It is easy to predict what the graph looks like. The code block given below creates a matplotlib window. The window shows a line passing through the given points. When the *plt.plot* is called, a Figure object is created, with axes, and finaly the data sets are plotted. Drawing a line example helps to understand how matplotlib works. Now let us try to draw a circle with building functions.

```
# -*- coding: utf-8 -*-
"""
Created on Fri May  5 14:47:46 2017

@author: fatih.cansu
Example of using circle patches
"""
import matplotlib.pyplot as plt
def build_circle():
    circle=plt.Circle((0,0), radius=0.4)
    return circle

def show_geo(patch):
    ax=plt.gca()#axis defining
    ax.add_patch(patch)#adding the axis figure
    plt.axis('scaled')#scaling the shape
    plt.show()#showing the shape
```

```
if __name__=='__main__':
    c=build_circle()
    show_geo(c)
```



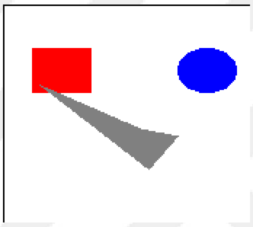
Besides creating axis and figure objects manualy, different functions in *pyplot* module can be used. When *gcf()* function is used, it returns a reference to the current *Figure*, and when we call *gca()* function, it returns that a reference to the current *Axes*. In this code block, program is seperated into two parts. Creation of *Circle* patch object and the addition of the patch to the figure with functions: *build_circle()* and *show_geo()*. In *build_circle()* a circle with radius and center coordinates is created. The *show_geo()* function is built such that it could work with any *mathplotlib* patches. The explanation of the *show_geo()* function was given on the code block with #. Furthermore, if you want to see the figure which is fitted to window you have to use *plt.axis('scaled')*. Because without this funciton the figure will be



Moreover, to take under guarantee the shape’s proportion to be 1:1, the *ax.set_aspect('equal')* could be used after *ax=plt.gca()*. You can also change the *edge color* and *face color* of the geometric shapes by using *fc='g'* returns green and *ec='r'* returns red.

Mathplotlib supports many geometric shapes such as *Ellipse*, *Polygon* and *Rectangle*. The other way of drawing geometric shapes is using the package *Pillow* (Sweigart, 2015). It is a bit simple than the *matplotlib*. Here is a given example.

```
from PIL import Image, ImageDraw
im=Image.new('RGBA', (170,150), 'white')
draw=ImageDraw.Draw(im)
draw.line([(0,0), (198,0), (198,198), (0,199), (0,1)],
fill='black')
draw.rectangle((20,30,60,60), fill='red')
draw.ellipse((120,30,160,60), fill='blue')
draw.polygon(((25,55), (94,85),
(120,90),(100,113)),fill='grey')

im.save('drawing.png')
```



3.3.2 Repeated Shapes, Fractals

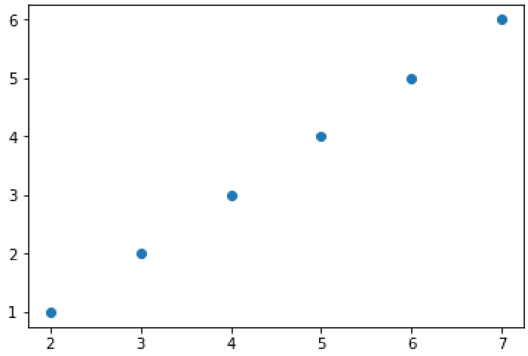
Fractals are interesting and complex geometric shapes which are constructed repeating simple geometric shapes. If we compare the fractals with other geometric shapes like circles, squares or any polygons, we will see that the fractals consist of infinite repetations. Infinite repetations of simple geometric shapes creates fractals, because if we look deeply, we can see that individual shapes repeated many times. Every simple shapes takes a little role of the huge construction like a brick on the *Great Wall*. Many of the fractals are constructed with the *geometric transformations* of the points or shapes. There are many computer programs to create the fractals but in this section we will discuss how to draw a fractal and what the construction algorithm is. And some popular examples such as *Barnsley Fern*, *The Sierpinski Triangle* and the *Henon Function* will be given.

3.3.3 Point Transformations

The main idea behind the construction a fractal is the transformation of a point. Let us assume that the point $A(x, y)$ is given as an initial point, the transformation be defined as $P(x, y) \rightarrow Q(x + 1, y + 1)$. This means that the location of the point will be changed by *one unit right* and *one unit up*. Let us write this simple transfomation.

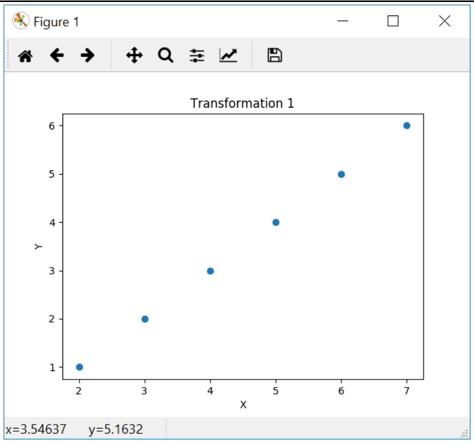
```
# -*- coding: utf-8 -*-
"""
Created on Sat May  6 14:56:28 2017

@author: fatih.cansu
"""
from pylab import plot, show
x0=2
y0=1
x_coordinates=[x0]
y_coordinates=[y0]

def transformation_x(x):
    return x+1
def transformation_y(y):
    return y+1

for i in range(0,5):
    x_coordinates.append(transformation_x(x0))
    y_coordinates.append(transformation_y(y0))
    x0=transformation_x(x0)
    y0=transformation_y(y0)

print(x_coordinates)
print(y_coordinates)
p=plot(x_coordinates,y_coordinates,'o'
```

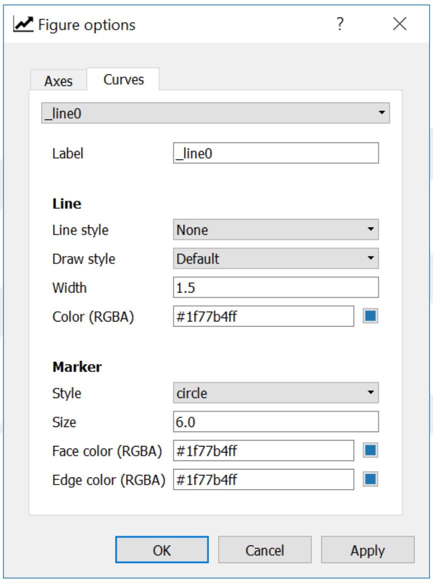
```
                      [2, 3, 4, 5, 6, 7]
                      [1, 2, 3, 4, 5, 6]
```



In the previous example, the *pylab* module was used. The *pylab* module is convenient for creating the plots from any given list, especialy, working on interactive Shell like IDLE Shell, as we have been doing many times so far. But if we are working in a big data list or we are writing for a larger program the *pyplot* module will be more efficient. There is no big differences for small code blocks, because all the methods that is given in *pylab* will work efficiently and the same way with *pyplot* and *using Anaconda IDLE*. Let’s convert the last example.

```
# -*- coding: utf-8 -*-
"""
Created on Sat May  6 14:56:28 2017

@author: fatih.cansu
"""

from matplotlib import pyplot
x0=2
y0=1
x_coordinates=[x0]
y_coordinates=[y0]

def transformation_x(x):
    return x+1
def transformation_y(y):
    return y+1

for i in range(0,5):
    x_coordinates.append(transformation_x(x0))
    y_coordinates.append(transformation_y(y0))
    x0=transformation_x(x0)
    y0=transformation_y(y0)

print(x_coordinates)
print(y_coordinates)
p=pyplot.plot(x_coordinates,y_coordinates,'o')
pyplot.show()
                      [2, 3, 4, 5, 6, 7]
                      [1, 2, 3, 4, 5, 6]
```





_line0

Label: _line0

Line

Line style: None

Draw style: Default

Width: 1.5

Color (RGBA): #1f77b4ff

Marker

Style: circle

Size: 6.0

Face color (RGBA): #1f77b4ff

Edge color (RGBA): #1f77b4ff

Figure 3.8. Figure options.

When the Anaconda IDLE is used, the output will be a bit different but useful. Because by using the *Options* panel one can change the *name of axis*, *label*, *legend*, *title* and etc. In the previous example the transformation of points was done by a single function. Let us assume that there exist more then one transformation function. And the transformation will be picked at randomly. The rules are given as

$$Transformation\ 1: P1(x, y) \rightarrow P2(x + 1, y - 1)$$

$$Transformation\ 2: P1(x, y) \rightarrow P2(x + 1, y + 2)$$

If we take the initial point as (0,1) then the new points will be

$$Transformation\ 1: P1(0,1) \rightarrow P2(1,0)$$

$$Transformation\ 2: P2(1,0) \rightarrow P3(2,2)$$

$$Transformation\ 2: P3(2,2) \rightarrow P4(3,4)$$

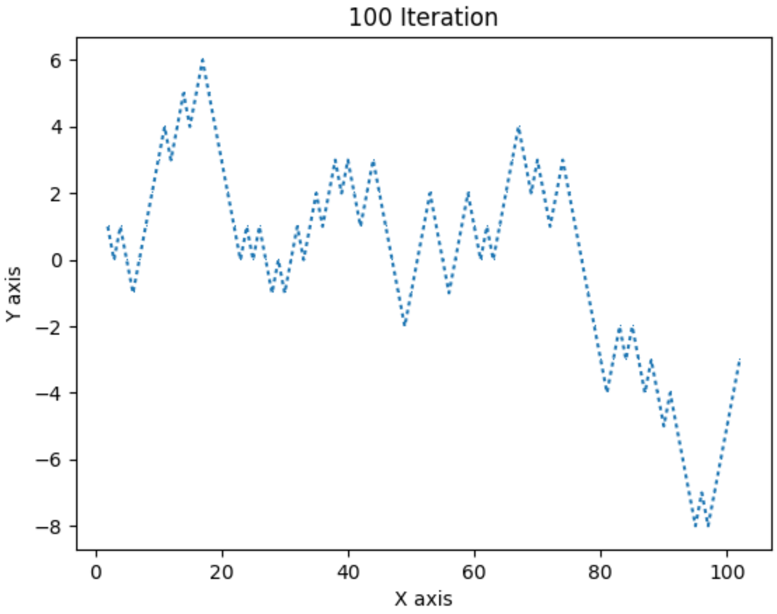
$$Tranformation\ 1: P4(3,4) \rightarrow P5(4,3)$$

$$\dots and\ so\ on.$$

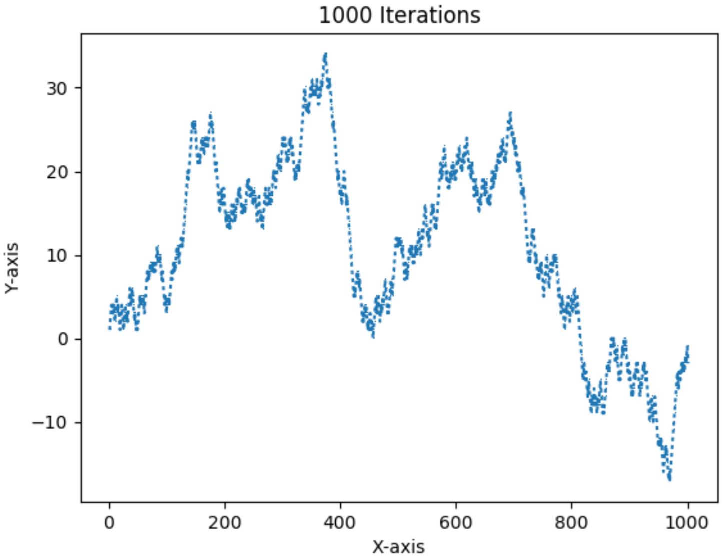
The selection of the transformations is done randomly. As seen from the transformations the point will follow a *zigzag* path. The following code block will draw a graph which consist the path of the initial point that is directed by transformations.

```
# -*- coding: utf-8 -*-
"""
Created on Sat May  6 14:56:28 2017

@author: fatih.cansu
"""

from matplotlib import pyplot
import random
x0=2
y0=1
x_coordinates=[x0]
y_coordinates=[y0]

def transformation1_x(x):
    return x+1
def transformation1_y(y):
    return y-1

def transformation2_x(x):
    return x+1
def transformation2_y(y):
    return y+1

for i in range(0,100):
    r=random.randint(0,1)
    if r==0:
        x_coordinates.append(transformation1_x(x0))
        y_coordinates.append(transformation1_y(y0))
        x0=transformation1_x(x0)
        y0=transformation1_y(y0)
    else:
        x_coordinates.append(transformation2_x(x0))
        y_coordinates.append(transformation2_y(y0))
```

```
        x0=transformation2_x(x0)
        y0=transformation2_y(y0)


p=pyplot.plot(x_coordinates,y_coordinates,'.')
pyplot.show()
```



If the iteration number is increased to 1000, the chart will be



Fractals are the geometric shapes that can be seen in nature like coastlines, trees and snowflakes. One of the popular fractals in nature which is invented and defined by English mathematician Michael Barnsley. The following steps are given by him to create fern like structure (Barnsley, 1988).

Transformation 1(0.85 probability):

$$x_{n+1} = 0.85x_n + 0.04y_n$$

$$y_{n+1} = -0.04y_n + 0.85y_n + 1.6$$

Transformation 2 (0.07 probability):

$$x_{n+1} = 0.02x_n - 0.26y_n$$

$$y_{n+1} = 0.23x_n + 0.22y_n + 1.6$$

Transformation 3 (0.07 probability):

$$x_{n+1} = -0.15x_n - 0.28x_n$$

$$y_{n+1} = 0.26x_n + 0.24y_n + 0.44$$

Transformation 4 (0.01 probability):

$$x_{n+1} = 0$$

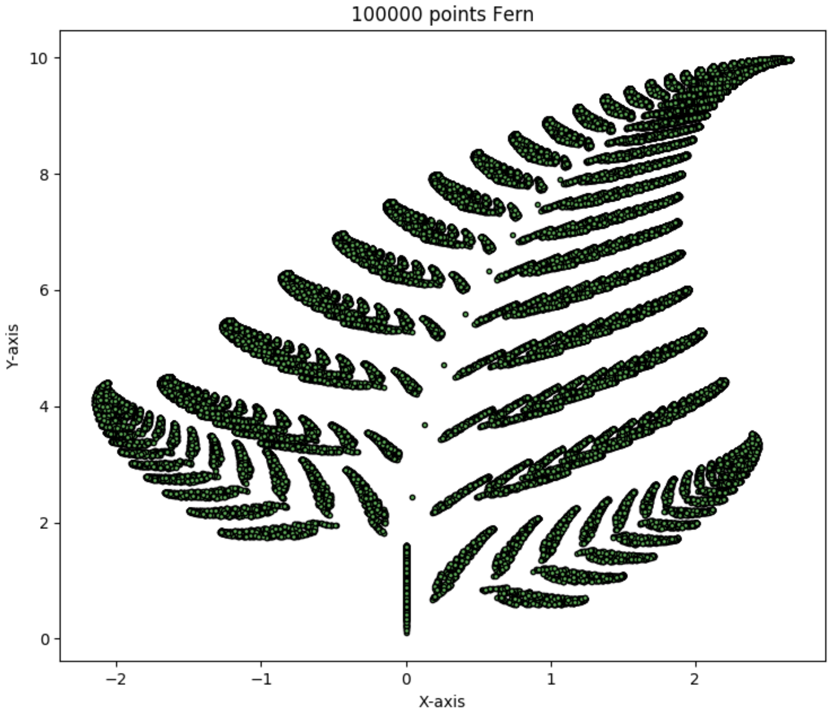
$$y_{n+1} = 0.16y_n.$$

Each of the given transformation creates a part of a fern. The first transformation which is selected with the *0.85* probability will creates the stem (root) and the bottom parts of the fern. The second and the third transformations will creates the bottom parts and left and, right respectively. At last, the fourth transformation will create the stem of the fern.

```
# -*- coding: utf-8 -*-
"""
Created on Sat May  6 14:56:28 2017

@author: fatih.cansu
Barns Fern Modelling
"""
```

```
from matplotlib import pyplot
import random
x0=0
y0=1
x_coordinates=[x0]
y_coordinates=[y0]

def transformation1_x(x,y):
    return 0.85*x+0.04*y
def transformation1_y(x,y):
    return -0.04*x+0.85*y+1.6

def transformation2_x(x,y):
    return 0.2*x-0.26*y
def transformation2_y(x,y):
    return 0.23*x+0.22*y+1.6

def transformation3_x(x,y):
    return -0.15*x+0.28*y
def transformation3_y(x,y):
    return 0.26*x+0.24*y+0.44

def transformation4_x(x,y):
    return 0
def transformation4_y(x,y):
    return 0.16*y
n=100
liste1=[]
liste2=[]
liste3=[]
liste4=[]
for i in range(1,int(n*0.85)+1):
    liste1.append(1)
for i in range(1,int(n*0.07)+1):
    liste2.append(2)
    liste3.append(3)

for i in range(1,int(n*0.01)+1):
    liste4.append(4)

for i in range(0,10**5):
        l=liste1+liste2+liste3+liste4
        r=random.choice(l)
        if r==1:
            x_coordinates.append(transformation1_x(x0,y0))
            y_coordinates.append(transformation1_y(x0,y0))
            x0=transformation1_x(x0,y0)
            y0=transformation1_y(x0,y0)
        elif r==2:
            x_coordinates.append(transformation2_x(x0,y0))
            y_coordinates.append(transformation2_y(x0,y0))
            x0=transformation2_x(x0,y0)
            y0=transformation2_y(x0,y0)
        elif r==3:
            x_coordinates.append(transformation3_x(x0,y0))
            y_coordinates.append(transformation3_y(x0,y0))
```

```
            x0=transformation3_x(x0,y0)
            y0=transformation3_y(x0,y0)
        else:
            x_coordinates.append(transformation4_x(x0,y0))
            y_coordinates.append(transformation4_y(x0,y0))
            x0=transformation1_x(x0,y0)
            y0=transformation1_y(x0,y0)



p=pyplot.plot(x_coordinates,y_coordinates,'.')
pyplot.show()
```



The transformations have different selection probabilities. For this reason, a non-uniform randomness in our code block has to used. Many of the fern modelling by using codeblocks consist an extra probability function. But instead of using probability functions, balls and box method is used. Let's assume that there exist 100 balls in a box and the number of blues are 85, whites and reds are 7 respectively and, purple is 1. The probability of taking a blue ball is 0.85 as expected. In our codeblock;

```
liste1=[]
liste2=[]
liste3=[]
liste4=[]
for i in range(1,int(n*0.85)+1):
    liste1.append(1)
```

```
for i in range(1,int(n*0.07)+1):
    liste2.append(2)
    liste3.append(3)
for i in range(1,int(n*0.01)+1):
    liste4.append(4)
l=liste1+liste2+liste3+liste4
```

creates a box *l=liste1+liste2+liste3+liste4* with different ball numbers and,

```
r=random.choice(l)
```

selects a ball non-uniformly.

As a second example, a new and popular fractal which was named by Polish mathematician Waclaw Sierpinski will be given. The Sierpinski fractal consist of equilateral triangles composed of smaller equilateral triangles. The transformations are given as

Transformation 1:

$$x_{n+1} = 0.5x_n$$

$$y_{n+1} = 0.5y_n$$

Transformation 2:

$$x_{n+1} = 0.5x_n + 0.5$$

$$y_{n+1} = 0.5y_n + 0.5$$

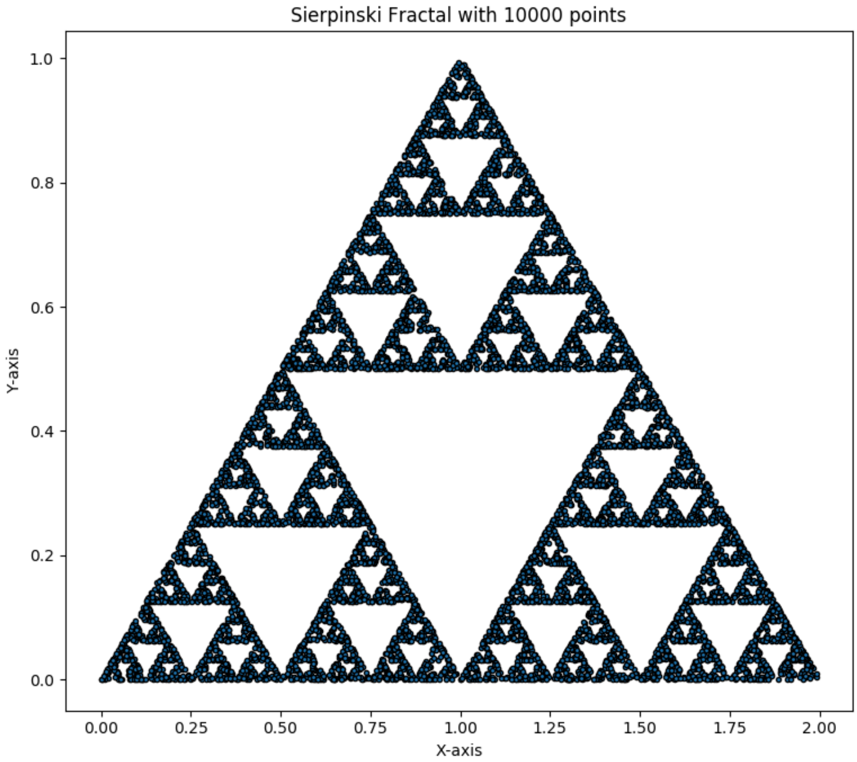
Transformation 3:

$$x_{n+1} = 0.5x_n + 1$$

$$y_{n+1} = 0.5y_n$$

Each of the transformations has the same selection probability. So there is no need to write a selection function or a balls and box codes as previous fractal.

```
# -*- coding: utf-8 -*-
"""
Created on Mon May  8 21:05:58 2017

@author: fatih.cansu
Sierpinski Triangle
"""
from matplotlib import pyplot
import random


def transformation1_x(x,y):
    return 0.5*x
def transformation1_y(x,y):
    return 0.5*y

def transformation2_x(x,y):
    return 0.5*x+0.5
def transformation2_y(x,y):
    return 0.5*y+0.5

def transformation3_x(x,y):
    return 0.5*x+1
def transformation3_y(x,y):
    return 0.5*y

def draw_Sierpinski(n):
    x0=0
    y0=0
    x_coordinates=[x0]
    y_coordinates=[y0]
    for i in range(0,n+1):
            l=[1,2,3]
            r=random.choice(l)
            if r==1:
                x_coordinates.append(transformation1_x(x0,y0))
                y_coordinates.append(transformation1_y(x0,y0))
                x0=transformation1_x(x0,y0)
                y0=transformation1_y(x0,y0)
            elif r==2:
                x_coordinates.append(transformation2_x(x0,y0))
                y_coordinates.append(transformation2_y(x0,y0))
                x0=transformation2_x(x0,y0)
                y0=transformation2_y(x0,y0)
            elif r==3:
                x_coordinates.append(transformation3_x(x0,y0))
                y_coordinates.append(transformation3_y(x0,y0))
                x0=transformation3_x(x0,y0)
                y0=transformation3_y(x0,y0)
    p=pyplot.plot(x_coordinates,y_coordinates,'.')
    return pyplot.show()
if __name__=='__main__':
    n=int(input('Enter the number of points: '))
    draw_Sierpinski(n)
```



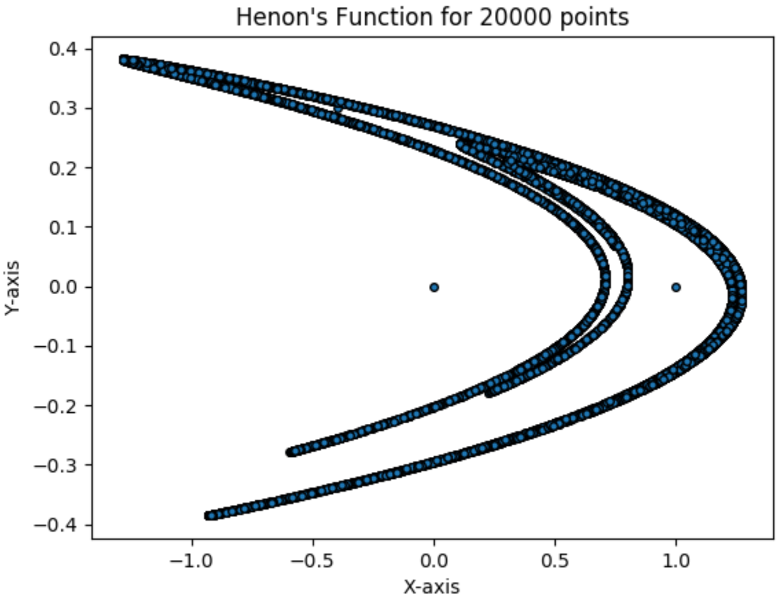
As a final example, we have another fractal which was introduced by Michel Henon at 1976. He invented a fuction which describes a transformation for a point as follows (Henon, 1976).

$$P(x,y) \rightarrow Q(y+1-1.4x^2, 0.3x).$$

Now, let us give the code for *Henon Function*.

```
# -*- coding: utf-8 -*-
"""
Created on Mon May  8 21:40:16 2017

@author: fatih.cansu
Henon Function
"""
from matplotlib import pyplot

def transformation(x,y):
    return y+1-1.4*x**2, 0.3*x

x0=0
y0=0
x_coordinates=[x0]
y_coordinates=[y0]
```

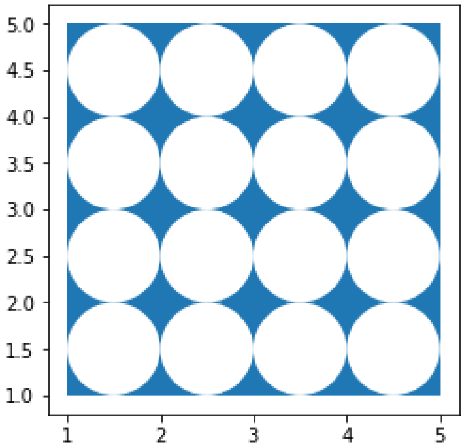
```
for i in range(0,20000):
    p=transformation(x0,y0)
    x_coordinates.append(p[0])
    y_coordinates.append(p[1])
    x0=p[0]
    y0=p[1]

pyplot.plot(x_coordinates,y_coordinates,'.')
pyplot.show()
```



In this section, it is started with how to draw a geometric shapes and draw a circle by using *mathplotlib* library. As well as, drawing a circle matplotlib allows drawing other geometric shapes. Now let’s draw a basic geometric shapes at the same coordinate axis:

```
# -*- coding: utf-8 -*-
"""
Created on Tue May  9 13:01:24 2017

@author: fatih.cansu
"""

import matplotlib.pyplot as plt
def build_square():
    square=plt.Polygon([(1,1),(5,1),(5,5),(1,5)], closed=True)
    return square
def build_circle(x,y):
```

```
    circle=plt.Circle((x,y), radius=0.5,fc='white')
    return circle

if __name__=='__main__':
    ax=plt.gca()#1.create the axis
    s=build_square()#2.create the square
    ax.add_patch(s)#3.add shape the axis
    y=1.5
    while y<5:
        x=1.5
        while x<5:
            c=build_circle(x,y) #4. create the circle
            ax.add_patch(c) #5. add shape the exist axis at step 3
            x=x+1
        y=y+1

    plt.axis('scaled')
    plt.show()
```



3.4 Problems and Solutions.

In this section, we will discuss and write the solution code of the given mathematics problems. The all the problems are taken from the on-line mathematics and programming challenge site *Project Euler (www.projecteuler.net)*. Every problem needs a basic mathematical knowledge and absolutely sharp algorithmic thinking because most of the solved problems are indeed *informatics olympiad problems*. Every problem at this chapter had been solved by the author of this thesis.

Problems are discussed in two parts. First part is understanding the mathematical pattern or generalization of the problem then the second part has the coding blocks. The degree of difficulty of every problem will follow an ascending order. The numerical answer of the every solution is given at last line of the codeblock with bold characters.

Before starting to solve the problems a library named *fkclib.py* is created to maket he codes run more efficiently. The library is given as

```
# -*- coding: utf-8 -*-
"""
Created on Tue May  9 20:35:07 2017

@author: fatih.cansu
The fkclib library
"""
import math
def lcm(a, b):
    return a * b // gcd(a, b)

def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
def prime_factors(n):##12=[2,2,3]
    i=2
    factors=[]
    while i*i<=n:
        if n%i!=0:
            i=i+1
        else:
            n//=i
            factors.append(i)
    if n>1:
        factors.append(n)
    return factors

def unique_prime_factors(n):##84=[2,3,7]
    i=2
    factors=[]
    while i*i<=n:
        if n%i!=0:
            i=i+1
        else:
            n//=i
            if i not in factors:
                factors.append(i)
    if n>1:
        if n not in factors:
            factors.append(n)
    return factors
```

```
def largest_prime_factor(n):##84=[7]
    i = 2
    while i * i <= n:
        if n % i:
            i += 1
        else:
            n //= i
    return n

def is_prime(n):
    if len(prime_factors(n))==1:
        return True
    else: return False

def list_of_divisors(n):
    liste=prime_factors(n)
    liste1=[]
    liste2=[]
    for elem in liste:
        if elem not in liste1:
            liste1.append(elem)
            liste2.append(liste.count(elem))
    return dict(zip(liste1, liste2))

def calculate(a,b):
    return (a**(b+1)-1)/(a-1)

def sum_of_divisors(n):
    toplam=1
    for k, v in list_of_divisors(n).items():
        toplam=toplam*calculate(k,v)
    return toplam-n

def number_of_divisors(n):
    value=1
    for v in list_of_divisors(n).values():
        value=value*(v+1)
    return value


def phi(n):
    liste=unique_prime_factors(n)
    value=1
    for elem in liste:
        value=(1-(1/elem))*value
    return int(n*value)

def prime_to_up(n):
    liste=[]
    for i in range(1,n+1):
        if is_prime(i)==True:
            liste.append(i)
    return liste


def first_n_prime(n):
```

```
    liste=[]
    i=0
    while len(liste)<n:
        if is_prime(i)==True:
            liste.append(i)
        i=i+1
    return liste

def is_palindrome(n):
    if str(n)[0::]==str(n)[::-1]:
        return True
    else: return False

def is_pandigital(n,r): #print(is_pandigital(123450,6))
    a=str(n)
    liste=[i for i in range(0,r+1)]
    k=0
    if len(a)!=r:
        return False
    else:
        for elem in liste:
            if str(elem) in str(n):
                k=k+1
            else:
                k=k
        if k==r:
            return True
        else: return False


def sqrt(x): # returns to floor value
       assert x >= 0
       i = 1
       while i * i <= x:
              i *= 2
       y = 0
       while i > 0:
              if (y + i)**2 <= x:
                     y += i
              i //= 2
       return y
def is_square(x):
       if x < 0:
              return False
       y = sqrt(x)
       return y * y == x
def reciprocal_mod(x, m):
       assert 0 <= x < m

       # Simplfied Euclide’s Algorithm by fatih cansu
       y = x
       x = m
       a = 0
       b = 1
       while y != 0:
              a, b = b, a - x // y * b
```

```
                x, y = y, x % y
        if x == 1:
                return a % m
        else:
                raise ValueError("Reciprocal does not exist")
```

Problem 1. The numbers which are divisible by 3 or 5 with zero remainder are listed. The first five elements are 3, 5, 6, 9 and 12. The sum of first five element could be calculated easily. So, calculate the sum of the numbers which are divisible by 3 or 5 and less than 1000.

Solution 1.

```
toplam=0
for i in range(0,1000):
    if i%3==0 or i%5==0:
        toplam=toplam+i
print(toplam)
                            233168
```

Problem 2. The elements of the Fibonacci sequence are obtained by adding last two elements each other. If we eliminate the first term of the sequence, we get 1, 2, 3, 5,… and it goes like this. According to this information calculate the sum of even terms of the Fibonacci series less than $4*10^6$.

Solution 2.

```
# -*- coding: utf-8 -*-
"""
Created on Tue May  9 20:22:15 2017

@author: fatih.cansu
prob2
"""
a=1
b=1
liste=[a,b]
toplam=0
while liste[-1]+liste[-2]<=4*10**6:
    elem=liste[-1]+liste[-2]
    liste.append(elem)
    if elem%2==0:
        toplam=toplam+elem

print(toplam)
                            4613732
```

Problem 3. The prime divisors of 44863 are 7, 13, 17 and 29. So find the greatest prime divisor of the number 2541876436891298753.

Solution 3.

```
import math
import time
start = time.time()
def div_num(n):
    div_list=[]
    for i in range(1,int(math.sqrt(n))+1):
        if n%i==0:
            div_list.append(i)
            div_list.append(n//i)
    return div_list

def is_prime(n):
    if len(div_num(n))==2:
        return True
    else:
        return False

def findprime(n):
    bigprime=0
    for number in div_num(n):
        if is_prime(number)==True and number>bigprime:
            bigprime=number
    return bigprime
print(findprime(2541876436891298753))
end=time.time()
print(end-start)
                                3924121
```

Problem 4. The numbers 11, 121 and 1441 have an interesting property such that the reverse sequence of digits of the each number is equal to itself. This type of numbers are also called as palindromic. So, find the greatest palindromic number which is equal to the product of two 3 digits numbers.

Solution 4.

```
import time
print(max(i*j for i in range(1000,100,-1) for j in
range(1000,100,-1)if str(i*j)==str(i*j)[::-1]))
sonra=time.time()
print("time:{}".format(str(sonra-once)[0:4]))
                                906609
```

Problem 5. The number 362880 is divisible by the each number (with zero reaminder) from 1 to 9. But we easily predict that the number is not the least number with this property. So, find the least number which is divisible by each number from 1 to 20.

Solution 5.

```
liste=[i for i in range(1,21)]

def lcm(a, b):
    return a * b // gcd(a, b)

def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
veri=1
for i in range(0,19):
    veri=lcm(veri, liste[i])

print(veri)
```

```
                                232792560
```

Problem 6. If we take the sum of the square of each number from 1 to 6 (namely $1^2+2^2+3^2+4^2+5^2+6^2$) it will be 91. But if we take the square of the sum of the numbers in same range ($(1+2+3+4+5+6)^2$) we will get 441. The difference is 441-91=350. So find the same difference for the numbers from 1 to 100.

Solution 6.

```
def square_sum(n):
    return n*(n+1)*(2*n+1)/6

def sum_square(n):
    return (n*(n+1)/2)**2

print(sum_square(100)-square sum(100))
```

```
                                25164150
```

Problem 7. It can easily seen that the 13 is the sixth prime number. Find the 10002 nd prime number?

Solution 7.

```
import math
import time
start = time.time()
def div_num(n):
    div_list=[]
    for i in range(1,int(math.sqrt(n))+1):
        if n%i==0:
            div_list.append(i)
    return div_list

def is_prime(n):
    if len(div_num(n))==1:
        return True
    else:
        return False

i=0
a=1
while i<=10001:
    if is_prime(a)==True:
        i=i+1
        prime=a
        a=a+1
    else:
        a=a+1
print(prime)
end = time.time()
print(end-start)
```

104759

Problem 8.

```
22251762402601583099503193927905505834569035731189
14143988264017605893005216239642128114116338574471
36029667534205191153166980258909663471706105828144
82812978708641406216112354014925771479780804072074
05969895977023934600609217478341331309942008344603
42331804443932726549512137440453444357600443085708
60067057897559988366110523151046419855243111334107
84206038247996800838326925469119763373793589749182
96002040278955499885018878041226420013610819309045
53980368479096042525646527940775929766651033355101
50734329895733294853990828508192678752495921701815
36518322048072349676109265353264838612716715863051
38047895222732073247700469234106405381105733643729
92311069552010017044247356146103424179733017728474
44077439980995734426608172780009793638619578653078
14464549914229093347411766833064140179134205630738
90702470741993866536256385453680195153359091084286
41139932618962515964369156846827606572541434974187
09218521916088328445808214381422717294719682715787
71069609319483591334680917676951369955901898190556
```

A table with 1000 digits is given. In the given table, the greatest product of 4 consequitive digit is 5832. So, find the greatest product of the 13 consecutive digits.

Solution 8.

```
def create_liste(filename):
    liste=''
    toplam=0
    with open(filename) as f:
        for line in f:
            liste=liste+str(line)
    return liste
numbers=[]
for elem in create_liste('prob8_numbers.txt'):
    if elem!='\n':
        numbers.append(elem)
    result=[]
for i in range(0, 1000-13):
    b=numbers[i:i+13]
    product=1
    for element in b:
        product=product*int(element)
    result.append(product)
print(max(result))
                                  49380710400
```

Problem 9. In a given right angled triangle which is not isosceles, the sum of three sides of the triangle is given as 1000. So find the product of the value of the sides.

Solution 9.

```
for a in range(1,998):
    for b in range(1,999-a):
        c = 1000 - a - b
        if a*a + b*b == c*c:
             print (a, b, c)
             print(a*b*c)
                                    31875000
```

Problem 10. The sum of prime numbers less than 15 is 2+3+5+7+11+13= 41. So find the sum of the prime numbers less than $2x10^6$.

Solution 10.

```
import math
import fkclib
toplam=0
for i in range(2,2000000):
```

```
    if fkclib.is_prime(i)==True:
        toplam=toplam+i
print(toplam)
```

```
142913828922
```

Problem 11. 20×20 table is given and four number in a diagonal painted red.

```
86 64 08 74 75 34 73 40 45 88 72 94 96 14 13 37 78 01 99 36
13 17 21 17 15 22 73 93 23 19 30 97 28 87 02 68 74 17 67 17
48 65 00 46 66 11 23 87 83 62 11 29 27 45 27 64 50 69 22 43
35 43 40 67 95 39 61 06 91 66 19 54 08 39 94 67 58 61 52 78
90 77 40 88 17 69 90 51 90 36 45 05 56 48 58 13 77 25 73 91
62 13 54 59 76 25 84 61 48 34 59 67 37 48 04 22 07 22 56 29
75 77 96 24 47 03 71 20 28 82 63 44 54 21 39 82 96 61 51 87
58 41 61 92 30 64 24 06 21 44 91 43 43 94 65 81 69 11 38 30
92 02 92 66 92 35 95 73 87 48 10 06 80 30 57 11 24 92 20 37
70 12 79 99 12 76 86 64 70 00 11 01 63 12 19 27 47 75 97 61
28 54 47 69 56 10 84 51 83 94 97 50 41 91 66 53 46 14 33 56
17 11 76 34 33 30 87 69 31 95 49 33 97 87 88 34 56 62 58 56
41 05 00 90 47 05 69 18 44 38 45 78 19 90 67 76 74 21 34 39
92 10 12 80 90 91 32 28 16 17 41 69 49 21 44 80 80 92 03 12
58 38 21 19 96 30 32 58 50 63 49 84 67 72 03 65 46 63 51 79
28 16 65 21 60 43 05 18 81 10 20 81 20 00 16 71 05 83 24 56
89 94 98 03 76 02 52 67 07 68 63 45 65 43 48 34 32 42 29 25
67 37 74 11 81 94 70 91 57 89 89 66 26 08 37 58 87 83 05 23
71 37 51 15 30 18 74 55 45 31 35 48 14 91 65 03 49 56 45 47
65 47 49 77 23 28 98 35 23 53 10 35 84 78 50 63 84 97 25 53
```

The product of 51 × 48 × 82 × 91 is calcuated as 18266976. So find the greatest production of four number which are on the same direction as up, left, right, down, or diagonally.

Solution 11.

```
def create_liste(filename):
    liste=[]
    toplam=0
    with open(filename) as f:
        for line in f:
            a=line.split()
            liste=liste+a
    return liste
numbers=[]
for elem in create_liste('numbers.txt'):
    numbers.append(float(elem))

results=[]
for i in range(0,400-20*3):
    a=numbers[i]
    b=numbers[i+20]
    c=numbers[i+40]
    d=numbers[i+60]
    results.append(a*b*c*d)

for j in range(0,400):
    if j%19!=0 or j%19!=18 and j%19!=17:
        a=numbers[j:j+4]
```

```
        sonuc=1
        for num in a:
            sonuc=sonuc*int(num)
        results.append(sonuc)

for k in range(0,399-21*3+1):
    if k%19!=0 or k%19!=18 and k%19!=17:
        e=numbers[k]
        f=numbers[k+21]
        g=numbers[k+42]
        h=numbers[k+63]
        results.append(int(e*f*g*h))
for k in range(0,399-19*3+1):
    if k%19!=1 and k%19!=2 and k%19!=3:
        e=numbers[k]
        f=numbers[k+19]
        g=numbers[k+38]
        h=numbers[k+57]
        results.append(int(e*f*g*h))
print(max(results))
```

```
61753344
```

Problem 12. A triangle number is obtained by summing the numbers up to a given number. For example 1+2+3+4=10 is the fourth triangle number. And the triangle number 28 has a unique property such that number of its divisors is greater than 5 and it is the first triangle number which has this property. Find the first triangle number whose number of divisors is over 500.

Solution 12.

```
import math
import time

def triangle_number(n):
    return int(n*(n+1)/2)

def DivNum(a):
    n=int(math.sqrt(a))
    div_list=[]
    for i in range(1, n+1):
        if a%i==0:
            if i and (a//i) not in div_list:
                div_list.append(i)
                div_list.append(a//i)
    return len(div_list)
once=time.time()
n=1
while DivNum(triangle_number(n))<500:
    n=n+1
```

```
sonra=time.time()
print(triangle_number(n))
print("time: {}".format(str(sonra-once)[0:4]))
                             76576500
                            time: 12.7
```

Problem 13. A table with 100 fifty-digit is given. Let’s assume that every line represents a number with 50 digits. If the sum of all the lines is given as A, find the first ten digit from left hand side of A.

```
88335047858190414612252130248050730063089607841393
81011069980616288250756773354051471152084117304907
06904408171012407035374783021478339974686886455591
55466544775544214641065001393660265487804328316959
00310954128440621673607846691274821555115298985864
39356291106537389930879311396779944986979375620763
51462141192476526016164179216903518572441376153960
65580681599618029366505839614687771718913940146692
78913881439358313479832358875048496127807646121222
84540190833675054233327435581039386752292179490527
72334856344980694959142476683217954774113735353638
25336413265360967360256395551717062483822430915898
12850907985264039334789208107947835871357542181145
84604118787545768893767016828771410791160542894245
48632305339217342929997382836483750716801961920278
72287373343887318356333640185404710426081627652984
12621548104067490802632565959964460572163103744574
18980211833143865640423405435818022560188751725854
98426749309347769784182901703937066919338638237356
74777949763986343695375281119434885434810031227912
57625964326267437806146606787330319363126626424935
15308670808747505640205233208887126325339044294879
08363042789230276296919346662540735578788602253628
19375411226595176452881706055665047831873054479743
10078808518498991231064045723987923580229409355785
79712565012005842071786508559555020269928919697359
66747508494770004310511917335040831649798691437344
33885747536139394158288273688249540465301975202601
04211866067459024072675058335167998375586671847723
04713168092494538491519732596651144248667790378031
38178074101446169094768022517777409445260064029098
48037540281315555919534024188993114235257203791810
55313965567694912198534593290351408528938929520446
08420620922644498812396893366059481480905591160194
16145828603548963422713655822868458139285664015567
72645152792843823108312004167551969100434619417722
74680906898089672366799772718563261583402781374306
20965105181830042310165159393521697107206554796537
04578256263659660938373161342856713701133707306681
62182397689143041411301759418853854945486425588983
42283885856293942374200181248594753471269847289394
24673740579330440609468319361732703304288164784274
03750473134582065936843646940672517058286915957594
27953304097681358206088990850674122007697347584191
98426472287462501694676721999635199138093896763395
12239212049776022524425819555354062392800817495309
10537802656704831618271579835152922893845636553533
37503976856466172972902403421003776195564553839549
23383235316660317440111615015308392109027310336354
21407010178709300832198024374378707505768328849411
75794455409644727347551054319712295844831774542008
26282091571455383054518117668942402223968877714946
91093460919569107395447082104651991473145831794734
42148301474480497696899455502350036010831967320984
23617556726029242708480975396170417713576227488860
58841812998502047729762737187196108072347069147930
60751224192982739043664184802086214125180452911878
65961455187626637561444164930646323572060415485236
```

```
16045678763851740833711763088061468627224204764178
42886381276513714065318356534669912073510307750884
17357041075986288601305967986964939722915230305260
08424859508819154204586528789322118248345862080656
11106995093624491225811910059345547343103920475752
18831173806773186723003253630092489127079992036920
35046881567834166824785876598578672592773169999970
08483229237034122376448033775552931394318329418374
19749581566214773809519608808495536531398429780820
34287776857168433728801524380927945978516169357059
61991422419427035937031341365985252982609976761117
57128086081214424967418399086530569661893356381959
21987140272666198458454951314632907076865521398586
09086573680313919087513638966124747477661331024133
91756500512778903493766056984557301752196424047875
26177236630079459572588699900577503780529594518207
41284309634032439345884539755248951605425784610285
89114498391774341446196638622412427777821599860988
75107965561744141708587072182531460395616078530494
11422880261918336140492092460017117793290045278185
65694350582460158842887957639409542516062051761048
84924416982612939067638983034912184348015430600429
48158189116492114490774620104966405189861586041921
89572453503064114946507426669410176854188163340522
74521903630825621414698788938944360960195409084156
10263358531695749570075240366886672780144030782351
79892362254220175110368711928660701939457497532762
06244723224220811465163056809768767664003559199492
39395378436072562531989984723021647180897150985968
06296012205491830973687243654208469139204276791607
53235458765239796662825545409759277282680112537466
45175067861239221413604467593886108105013559903756
77009123815777570871949862927846879203962435553138
86089412073539787794523906014959354943875499449686
77190912807024127635814530992822922323871344400777
10606616039659017379847901322053062548169109750139
19224005548754417301630412981107284249060650247373
71388078763693247655725112422179606050707236270960
34105065408703479564414916410423154262785456824767
11147524308379824390557243783624772952430928168341
91675106935462269280985304433648211334407437745379
91626970306159130233206764127533738529332197628799
```

Solution 13.

```
with open('prob13.txt', 'r') as f:
    liste = [line.strip() for line in f]#Every line in txt file added
#to list
toplam=0
for i in range(len(liste)):
    toplam=toplam+int(liste[i])

a=str(toplam)
print(a[0:10])
                                4391266421
```

Problem 14. Collatz sequence is a famous number sequence of mathematics. Every element of the sequence is obtained by using a simple algorithm. For a given number if the number is even the next number will be half of the previous, else the next number is equal to three times and one more of the previous. The mystery is starts at this point because the algorithm is ended always by 1. For example if we take number as 7:

7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 6, 4, 2, 1

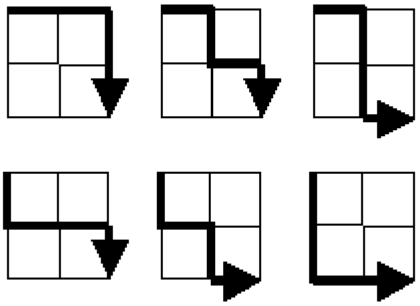
So by using the given rules calculate the Collatz number less than one milllion which reaches the number 1 with the maximum step number.

Solution 14.

```
def collatz_len(n):
    i=1
    while n!=1:
        if n%2==0:
            n=n/2
            i=i+1
        else:
            n=3*n+1
            i=i+1
    return i

maxi=0
maxlen=0
for i in range(1, 1000001):
    if maxlen<=collatz_len(i):
        maxlen=collatz_len(i)
        maxi=i

print(maxi)
```

```
                            837799
```

Problem 15.



The directions on a 2x2 map is given as six different routes (only right and down move). Since the map is so small all the routes are predictable. So find the number of routes from left-up corner to right-bottom corner on a 20x20 map.

Solution 15.

```
def fact(n):
    fac=1
    for i in range(1,n+1):
        fac=fac*i
    return fac

def route(n,r):
    return(int(fact(2*n)/((fact(r)*fact(r)))))
print(route(20,20))
                        137846528820
```

Problem 16. 2^{16}=65536 is given and and its sum of the digits is 6+5+5+3+6=25. So find the digit sum of the number 2^{1000}.

Solution 16.

```
a=2**1000

toplam=0
for char in str(a):
    toplam=toplam+int(char)
print(toplam)
                            1366
```

Problem 17. The factorial is a well-known mathematical expression. So find the sum of digits of 100!.

Solution 17.

```
def fact(n):
    fact=1
    for i in range(1,n+1):
        fact=fact*i
    return fact
a=str(fact(100))
sum=0
for number in a:
    sum=sum+int(number)

print(sum)

                             648
```

Problem 18. Let's assume that s(n) represents the sum of the divisors (less than and different than n) of the n. If s(x)=y and s(y)=x where $x \neq y$ then (x,y) is called as

Sbelian pairs. For example (220,284) are *Sbelian* pairs. Sum of the divisors of 220 is 284 and sum of the divisors of the 284 is 220. So, find the sum of all *Sbelian* numbers under 10000.

Solution 18.

```
import math

def divsum(a):
    n=int(math.sqrt(a))
    div_list=[]
    for i in range(1, n+1):
        if a%i==0:
            if i not in div_list:
                div_list.append(i)
            if a//i not in div_list:
                div_list.append(a//i)
    div_list.sort()
    return sum(div_list)-a
toplam=0
for i in range(1,10001):
    a=divsum(i)
    if divsum(a)==i and a!=i:
        toplam=toplam+a
print(toplam)
                             31626
```

Problem 19. Given names.txt [1] includes english names over 5000 which are sorted into alphabetical order. Every letter has a numerical value which is the position number. For example a=1, b=2, c=3 and goes on. So, every name has a numerical value for example F+A+T+I+H=6+1+18+9+8=42. The position number of FATIH is is 23 in *names.txt* hence the score of FATIH is 42x23=966. Calculate the score of the every name in names.txt and find the sum of the score of the all names in file.

Solution 19.

```
import time
then=time.clock()
liste=[chr(i) for i in range(ord('A'),ord('Z')+1)]
liste.append('"')
open_file = open('p022_names.txt')
lst = [] #empty list
```

---

[1] Download the file names.txt from the address
https://drive.google.com/open?id=0B5QoqCRDwQR3NXhjUFNfOU9ERTg

```
for line in open_file:
    line = line.rstrip()   # We aligned a new line to the line
according to the leading and trailing space.
    words = line.split()   # We added word by word all words
to a new list according to the gap between the resulting
lines.
    for word in words:
        lst.append(word)
lst.sort()
newlist=""
for i in range(len(lst)):
    newlist=newlist+lst[i]
wordliste=[word for word in newlist.split(",")]
wordliste.sort()

def kelimator(wordliste):
    wordnum=[]
    for word in wordliste:
        toplam=0
        for char in word:
            toplam=toplam+(liste.index(char)+1)
        toplam=toplam-(54)
        wordnum.append(toplam*(wordliste.index(word)+1))
    return sum(wordnum)

print(kelimator(wordliste))
now=time.clock()
print(now-then)
```

Solution 19 (Alternative).

```
f = open('p022_names.txt', 'r')
total = 0
for k, name in enumerate(sorted(f.read().replace("\"",
"").split(","))):
        points = 0
        for char in list(name):
                points += ord(char)-64
        total += points * (k+1)
print(total)
                           871198282
```

Problem 20. All the possible permutations of 3,1,2,4 is calculated. The number of the all permutations without repetation is 4!=24. For example if we order the set first element will be 1234 and the last element will be 4321. Due to this permutation rules, find the millionth permutations of the 0,1,2,3,4,5,6,7,8,9. (note: 0567894321 is a possible permutation.)

Solution 20.

```
import math
import itertools
a=1000000
liste=[0,1,2,3,4,5,6,7,8,9]
nliste=[]
for i in range(len(liste)-2):
    kalan=a%(math.factorial(len(liste)-1))
    if kalan!=0:
        bolum=a//(math.factorial(len(liste)-1))
        nliste.append(liste[bolum])
        a=kalan
    else:
        a=2
        bolum=a//(math.factorial(len(liste)-1))
        nliste.append(liste[bolum])

    del liste[bolum]
liste.sort(reverse=True)
print(nliste+liste)
```

```
                          2783915460
```

Problem 21. The Fibonacci sequence has an interesting property such that

$$f_n = f_{n-1} + f_{n-2} \text{ where } f_1 = 1 \text{ and } f_2 = 1.$$

The 144 is the 12th fibonacci number and its is the first fibonacci number wtih 3 digits. Find the index of the first fibonacci number which contains 1000 digits.

Solution 21.

```
def fibo(n):
    a,b=1,1
    for i in range(1,n):
        a,b=b, a+b
    return a
a=1
while len(str(fibo(a)))<1000:
    a=a+1
print(a)
```

```
                             4782
```

Problem 22. : If we take a fraction with a numerator equal to one and denominators range from 2 to 10, then the fractions and their decimal representations would be as given:

1/2 = 0.5

1/3 = $0.\overline{3}$

1/4 = 0.25

1/5 = 0.2

1/6 = $0.1\overline{6}$

1/7 = $0.\overline{142857}$

1/8 = 0.125

1/9 = $0.\overline{1}$

1/10 = 0.1

From the table given above it is clear that the length of the repeating part of the decimals change in every fraction. For example the length of repeating part of the fraction 1/7 is 6. So, find the denominator of the decimal whose repeating part is the longest and whose denominator is less than 1000.

Solution 22.

```
def ind(n):
    x=n
    while x%5==0:
        x=(x/5)
    y=x
    while y%2==0:
        y=(y/2)
    return int(y)

def g(n,d):
    d=ind(d)
    a=1
    num=10*n
    while (num-n)%d:
        num=num*10
        a=a+1
    return a


maxper=0
for i in range(1,1001):
    if maxper<g(1,i):
        maxper=g(1,i)
```

```
        a=i
print(maxper,a)
                                            982,983
```

Problem 23. The great mathematician Leonard Euler discovered a formula:

$$n^2 + n + 41\,.$$

This formula has an interesting property such that for n value if the numbers from 1 to 39 are substituted then formula generates 40 prime numbers. By using substitution it can easily found. Then the marvellous formula was discovered such that it produces 80 prime numbers by substitution values of n:

$$n^2 - 79n + 1601, 0 \leq n \leq 79.$$

Let’s assume that the formula is given:

$$n^2 + f \cdot n + c, where\ |f|, |c| < 1000.$$

Find the value of $f \cdot c$ such that the formula generates maximum prime numbers for successive values of n (the initial value of n=0).

Solution 23.

```
import fkclib
import time
start=time.time()
liste=[]
vliste=[]
maxivalue=0
for a in range(-999,1000):
    for b in range(-1000,1001):
        value=0
        step=0
        while fkclib.is_prime(step**2+a*step+b):
            step=step+1
            value=value+1
        liste.append(a*b)
        vliste.append(value)
print(max(vliste))
a=vliste.index(max(vliste))
print(liste[a])
end=time.time()
print("time: {}".format(end-start))
                                            71
```

```
-59231
time: 24.5
```

Problem 24.

21	22	23	24	**25**
20	**7**	8	**9**	10
19	6	**1**	2	11
18	**5**	4	**3**	12
17	16	15	14	**13**

A 5x5 table which filled numbers is given and the numbers on the diagonals colorized with red. For the given table, sum of the colorized numbers can be easily calculated. Calculate the sum of the numbers on the diagonals for 1001 by 1001 table.

Solution 24.

```
a=2
value=1
toplam=1
for j in range(1,501):
    for i in range(1,5):
        value=value+a
        toplam=toplam+value
    a=a+2
print(toplam)
```

```
669 171 001
```

Problem 25. All combinations for a^b for $2 \leq a \leq 5$ and $2 \leq b \leq 5$ is given as:

$2^2=4$, $2^3=8$, $2^4=16$, $2^5=32$
$3^2=9$, $3^3=27$, $3^4=81$, $3^5=243$
$4^2=16$, $4^3=64$, $4^4=256$, $4^5=1024$
$5^2=25$, $5^3=125$, $5^4=625$, $5^5=3125$

If the numbers are ordered then the sequence will be:

4, 8, 9, 16, 25, 27, 32, 64, 81, 125, 243, 256, 625, 1024, 3125

If a^b for $2 \leq a \leq 100$ and $2 \leq b \leq 100$ are given, calculate the number of distinct elements in the obtained sequence?

Solution 25.

```
list=[]
for i in range(2,101):
    for j in range(2,101):
        if i**j not in list:
            list.append(i**j)
print(len(list))

                             9183
```

Problem 26. There is only 3 four digit numbers which can be written as the sum of the fourth power of its digits:

$$1634 = 1^4 + 6^4 + 3^4 + 4^4$$

$$8208 = 8^4 + 2^4 + 0^4 + 8^4$$

$$9474 = 9^4 + 4^4 + 7^4 + 4^4$$

The sum of 1634 + 8208 + 9474 is equal to 19316. So, find the sum of all numbers which can be written as the sum of fifth power of its digits.

Solution 26.

```
def powsum(n):
    a=str(n)
    toplam=0
    for digit in a:
        toplam=toplam+(int(digit)**5)
    if (n==toplam):
        return True
    else:
        return False
toplam=0
for i in range(4149,1000000):
    if powsum(i)==True:
        toplam=toplam+i
print(toplam)
                             443839
```

Problem 27. For an n-digit number, if the digits of number consist of all the integers 1 to n, it is called as pandigital number. For example 23415 is a five digit pandigital. Even more, some numbers have very interesting identities about being pandigital. For example the number 7254 is the product of the numbers 39 and 186. The digits of two factors and the result consist of 1 to 9 numbers. Find the sum of all numbers whose two factors and itself have all numbers 1 to 9.

Solution 27.

```
import math
def DivNum(a):
    n=int(math.sqrt(a))
    div_list=[]
    for i in range(1, n+1):
        if a%i==0:
            if i and (a//i) not in div_list:
                div_list.append(i)
                div_list.append(int(a//i))
    return div_list

def kontrol(n):
    a=str(n)
    liste=[str(i)for i in range(1,10)]
    kontrolliste=[]
    for element in liste:
        if element in a:
            kontrolliste.append(1)
    if len(kontrolliste)==9:
        return True
    else:
        return False

def pandigital(n):
    checklist=[]
    for element in DivNum(n):
        a=str(n)+str(int(n//element))+str(element)
        if kontrol(a)==True and len(a)==9:
            checklist.append(1)
    if 1 in checklist:
        return True
    else:
        return False
total=0
for i in range(1,50000):
    if pandigital(i)==True:
        total=total+int(i)
print(total)
                              45228
```

Problem 28. The fraction 49/98 has an interesting property. If we cancel the same digit in the nominator and denominator 49/98=4/8=1/2 and it is surprisingly true. Mathematicaly, doing cancellation by this way is not true but the result is true. 40/20, 30/50, 70/80 are the trivial solutions but there exist four non-trivial solutions. Find the denominator of product of these non-trivial solutions.

Solution 28.

```
import math

for i in range(10,100):
    for j in range(10,100):
        pay=str(i)
        payda=str(j)
        try:
                if pay[0]==payda[0]:
                    if int(pay[1])/int(payda[1])==i/j:
                        if i!=j and i*j%100!=0:
                            print(i,j)
                if pay[0]==payda[1]:
                    if int(pay[1])/int(payda[0])==i/j:
                        if i!=j and i*j%100!=0:
                            print(i,j)
                if pay[1]==payda[0]:
                    if int(pay[0])/int(payda[1])==i/j:
                        if i!=j and i*j%100!=0:
                            print(i,j)
                if pay[1]==payda[1]:
                    if int(pay[0])/int(payda[0])==i/j:
                        if i!=j and i*j%100!=0:
                            print(i,j)
        except ZeroDivisionError:
                pass
```

```
16 64
19 95
26 65
49 98
64 16
65 26
95 19
98 49
answer: 100
```

Problem 29. 145 is a very interesting number because the sum of the factorial of its digits is equal to itself. Namely, 1!+4!+5! is equal to 145. Find the sum of all numbers that keep the same manner.(1 and 2 not included.)

Solution 29.

```
def fact(n):
    if n==0:
        return 1
    else:
        fact=1
        for i in range(1,n+1):
            fact=fact*i
        return fact
```

```
liste=[]
for i in range(3,100000):
    toplam=0
    for j in str(i):
        toplam=toplam+fact(int(j))
        if toplam==i:
            liste.append(i)
toplam=0
for num in liste:
    toplam+=num
print(toplam)
                             40730
```

Problem 30. The prime numbers are the big phenomenon of the mathematics. For example 971 is a prime number. Even more, all circulations of the number is also prime number as 197 and 719. There exist thirteen primes which provide the circle prime rule. For example 5,7,71,37. So, find the number of circular primes less than 10^6.

Solution 30.

```
import math
import fkclib

def rotation(n):
    liste=[n]
    a=str(n)
    for i in range(1,len(a)):
        number=a[1:]+a[0]
        liste.append(int(number))
        a=str(number)
    b=0
    for num in liste:
        if fkclib.is_prime(num)==True:
            b=b+1
    if b==len(a):
        return True
    else: return False
c=0
for i in range(1,10**6,1):
            if '2' not in str(i):
                if fkclib.is_prime(i)==True:
                    if rotation(i)==True:
                        c=c+1

print(c+1)
                              55
```

Problem 31. If a number read from right to left and left to right is equal each other, called as palindromic numbers. For example 1221 is an palindromic number. 585 is also a palindromic number. Even more, the binary form of the number 585 is also palindromic: $(1001001001)_2$. Calculate the sum of all palindromic numbers less than 10^6 whose binary expansion is also palindromic.

Solution 31.

```
def div(m,n):
    i=0
    while (m>=n)==True:
        m=m-n
        i=i+1
    return i

def base_conv(m,n):
    converted=""
    while(m>=n)==True:
        number=m%n
        converted=str(number)+converted
        m=div(m,n)
        if m<n:
            converted=str(m)+converted
    return converted

def is_palindrome(n):
    a=str(n)
    if (a==a[::-1])==True:
        return True
    else:
        return False

liste=[i for i in range(1,1000000) if is_palindrome(i)==True]

toplam=0
for elem in liste:
        if is_palindrome(base_conv(elem,2))==True:
            toplam=toplam+elem
print(toplam)
```

```
872187
```

Problem 32. The number 3137 has an amazing property such that the numbers 3137, 137, 37 and 7 are all primes. Moreover, 3137, 313, 31 and 3 are all also prime numbers. Let’s call the number 3137 as *Bâde Number*. Find the first eleven Bâde Numbers and their sum.(note: pimes less than 8 not accepted as *Bâde Number*.)

Solution 32.

```
import math
import fkclib_alternative


def controlfromleft(n):
    checklist=[]
    a=str(n)
    for i in range(len(a)+1):
        b=n//(10**(len(a)-i))
        if fkclib_alternative.is_prime(b)==True:
            checklist.append(1)
    if len(checklist)==len(a):
        return True
    else:
        return False

def controlfromright(n):
    checklist=[]
    a=str(n)
    for i in range(len(a)+1):
        b=n%(10**(len(a)-i))
        if fkclib_alternative.is_prime(b)==True:
            checklist.append(1)
    if len(checklist)==len(a):
        return True
    else:
        return False
pliste=["2","3","5","7"]
toplam=0
count=0
i=8
while count<11:
    if str(i)[-1] in pliste and str(i)[0] in pliste:
        if controlfromleft(i)==True and
controlfromright(i)==True:
            print(i)
            toplam=toplam+i
            count=count+1
            i=i+1
        else: i=i+1
    else: i=i+1

print(toplam)
```

```
                              23
                              37
                              53
                              73
                             313
                             317
                             373
                             797
                             3137
                             3797
```

```
739397
748317
```

Problem 33. Take the number 192 and multiply it by each of 1, 2, and 3:

$$192 \times 1 = 192$$

$$192 \times 2 = 384$$

$$192 \times 3 = 576$$

By concatenating each product we get the 1 to 9 pandigital, 192384576. We will call 192384576 the concatenated product of 192 and (1,2,3). The same can be achieved by starting with 9 and multiplying by 1, 2, 3, 4, and 5, giving the pandigital, 918273645, which is the concatenated product of 9 and (1,2,3,4,5). What is the largest 1 to 9 pandigital 9-digits number that can be formed as the concatenated product of an integer with (1,2, ... , n) where $n > 1$?

Solution 33.

```
def check(a):
    return sorted(str(a))==sorted(str(123456789))
def prod(a):
    i=1
    num=''
    while len(num)<9:
        num=num+str(a*i)
        i=i+1
    if check(num)==True:
        return num
    else: return 0
numlist=[]
panlist=[]
for i in range(1,10000):
    if prod(i)!=0:
        numlist.append(i)
        panlist.append(prod(i))
print(dict(zip(numlist,panlist)))
print(len(numlist))
print(max(panlist))
{192: '192384576', 1: '123456789', 7269: '726914538', 327:
'327654981', 6792: '679213584', 9: '918273645', 9327:
'932718654', 7692: '769215384', 6927: '692713854', 273:
'273546819', 9267: '926718534', 7923: '792315846', 7329:
'732914658', 6729: '672913458', 9273: '927318546', 219:
'219438657', 7932: '793215864', 7293: '729314586'}
18
932718654
```

Problem 34. Let’s assume that for a given Pythagorean triple {x,y,z} the sum of them be equal to p. For p=120, we have three triple (24,45,51), (30,40,50) and, (20,48,52). Find the value of p ≤ 1000 such that the number of triples is maximum.

Solution 34.

```
import math
def is_square(n):
    return math.sqrt(n)==int(math.sqrt(n))
liste=[]
for a in range(1,1000):
    for b in range(1,1000):
        c=a**2+b**2
        if is_square(c)==True and a+b+math.sqrt(c)<=1000:
            liste.append(a+b+math.sqrt(c))

liste.sort()
def findhighfreq(liste):
    numlist=[]
    flist=[]
    for i in range(len(liste)):
        j=0
        while liste[i]!=liste[j]:
            j=j+1
        numlist.append(liste[i])
        flist.append(i-j+1)
    a=flist.index(max(flist))
    b=numlist[a]
    return b

print(findhighfreq(liste))
```

```
                              840
```

Problem 35. The number N is created by concatenating the numbers from 1 to n. Decimal representation of the number is N=0.123456789**1**011… . It is easily seen that the the 10 th digit is 1. If N(i) represents the i’th digit in the N find the product:

$$N(1) \times N(10) \times N(100) \times N(1000) \times N(10000) \times N(100000) \times N(1000000)$$

Solution 35.

```
def create():
    num='0'
    for i in range(1,179000):
        num=num+str(i)
    return num
liste=create()
print(len(liste))
```

```
carpim=1
for i in range(0,7):
    carpim=carpim*int(liste[10**i])

print(carpim)
                                  210
```

Problem 36. A pandigital number is an n-digit number and consists all numbers from 1 to n in its digits. For example 3124 is a four digits pandigital number. Find the largest pandigital number which is also a prime number.

Solution 36.

```
import math
import fkclib
def is_pandigital(n):
    liste=[]
    a=str(n)
    for i in range(1,len(a)+1):
        if str(i) in a:
            liste.append(1)
    if len(liste)==len(a):
        return True
    else: return False

for i in range(1,987654322):
    if is_pandigital(i)==True:
        if fkclib.is_prime(i)==True:
            print(i)

                                  7652413
```

Problem 37. The triangle numbers are given with the closed form $t_n = \frac{n(n+1)}{2}$. The first five triangle numbers are 1, 3, 6, 10, and this goes on. The file words.txt[2] includes more than 2000 words. The value of every word is calculated with a special method. Due to method, the value of a word is equal to the letter number of each word in the alphabetical order of the English language. For example ZEYNEP=Z+E+Y+N+E+P=26+5+25+15+5+16=87. If the value of the word is a triangle number the word is called as triangle word. So, find the number of triangle words in file words.txt.

---

[2] Download the file names.txt from the address https://drive.google.com/file/d/0B5QoqCRDwQR3d19hNVhhVnhYdjg/view?usp=sharing

Solution 37.

```
import time
liste=[chr(i) for i in range(ord('A'),ord('Z')+1)]
liste.append('"')
open_file = open('p042_words.txt')
lst = [] #boÅŸ liste

for line in open_file:
    line = line.rstrip()
    words = line.split()
    for word in words:
        lst.append(word)
lst.sort()

newlist=""
for i in range(len(lst)):
    newlist=newlist+lst[i]
wordliste=[word for word in newlist.split(",")]

def is_triangle(n):
    if ((1+8*n)**(0.5)).is_integer():
        return True
    else:
        return False
def is_triangle_word(word):
    toplam=0
    for char in word:
        toplam=toplam+liste.index(char)+1
    toplam=toplam-(2*(liste.index('"')+1))
    if is_triangle(toplam)==True:
        return True
    else:
        return False
starttime=time.clock()
a=0
for i in range(len(wordliste)):
    if is_triangle_word(wordliste[i])==True:
       a=a+1
print(a)
endtime=time.clock()
print(endtime-starttime)
                              162
                          0.13 second
```

Problem 38. The number, 1406357289, is a 0 to 9 pandigital number because it is made up of each of the digits from 0 to 9 in some order, but it also has a rather interesting sub-string divisibility property. Let d_1 be the 1st digit, d_2 be the 2nd digit, and so on. In this way, we note the following:

$d_2d_3d_4$=406 is divisible by 2

$d_3d_4d_5$=063 is divisible by 3

$d_4d_5d_6$=635 is divisible by 5

$d_5d_6d_7$=357 is divisible by 7

$d_6d_7d_8$=572 is divisible by 11

$d_7d_8d_9$=728 is divisible by 13

$d_8d_9d_{10}$=289 is divisible by 17

Find the sum of all 0 to 9 pandigital numbers with this property.

Solution 38.

```
def is_pandigital(num):
    liste=[str(i) for i in range(0,10)]
    checklist=[]
    for char in num:
        if char in liste:
            checklist.append(1)
    if sum(checklist)==10:
        return True
    else:
        return False

def divisors(n):
    divlist=[1,2,3,5,7,11,13,17]
    value=[]
    for i in range(1,8):
        if int(n[i:i+3])%divlist[i]==0:
            value.append(1)
    if sum(value)==7:
        return True
    else:
        return False


from itertools import permutations
l = list(permutations(range(0, 10)))

newL=[]
for element in l:
    kelime=''
    for i in range(len(element)):
        kelime=kelime+str(element[i])
    newL.append(kelime)
```

```
def sum_pan():
    toplam=0
    for element in newL:
        if is_pandigital(element)==True and
divisors(element)==True:
                toplam=toplam+int(element)
    return toplam
print(sum_pan())
                         16695334890
```

Problem 39. Pentagonal numbers are generated by the formula, $P_n=n(3n-1)/2$. The first ten pentagonal numbers are:

1, 5, 12, 22, 35, 51, 70, 92, 117, 145, ...

It can be seen that $P_4 + P_7 = 22 + 70 = 92 = P_8$. However, their difference, $70 - 22 = 48$, is not pentagonal. Find the pair of pentagonal numbers, P_j and P_k, for which their sum and difference are pentagonal and $D = |P_k - P_j|$ is minimised; what is the value of D?

Solution 39.

```
def is_pentagonal(n):
    if math.sqrt(24*n+1)==int(math.sqrt(24*n+1)):
        return True
    else:
        return False

def pentagonal(n):
    return int(n*(3*n-1)*(0.5))

pentafark=[]
pentaliste=[pentagonal(i) for i in range(1,10000)]
for i in range(0,len(pentaliste)):
    for j in range(i-1,0,-1):
            if
is_pentagonal(pentaliste[i]+pentaliste[j])==True and
is_pentagonal(pentaliste[i]-pentaliste[j])==True:
                       print(pentaliste[i]-pentaliste[j],
pentaliste[i], pentaliste[j])

(1247, 715, 532)
(2262, 1820, 442)
(12927, 7315, 5612)
(25676, 23375, 2301)
```

```
(73151, 12650, 60501)
(661012, 490490, 170522)
(3079517, 2794155, 285362)
(3455727, 270725, 3185002)
(7042750, 1560090, 5482660)
                            5482660
```

Problem 40. The general closed formula of the pentagonal, triangle and hexagonal numbers:

Triangle Number Closed Formula	$T_n=n(n+1)/2$	1, 3, 6, 10, 15, ...
Pentagonal Number Closed Formula	$P_n=n(3n-1)/2$	1, 5, 12, 22, 35, ...
Hexagonal Number Closed Formula	$H_n=n(2n-1)$	1, 6, 15, 28, 45, ...

The number 40755 has an interesting property that is triangle, pentagonal and also hexagonal number. So, let's call 40755 as *Zeynep Number*. Find the next Zeynep number greater than 40755.

Solution 40.

```
import math
import time
def is_triangle(n):
    if math.sqrt(1+8*n)==int(math.sqrt(1+8*n)):
        return True
    else: return False
def is_pentagonal(n):
    if (math.sqrt(1+24*n)+1)/6==int((math.sqrt(1+24*n)+1)/6):
        return True
    else: return False
def is_hexagonal(n):
    if
(math.sqrt(1+8*n)+1)*(1/4)==int((math.sqrt(1+8*n)+1)*(1/4)):
        return True
    else: return False
def tri(n):
    trilist=[]
    for i in range(1,n+1):
        trilist.append(int(i*(i+1)*(0.5)))
    return trilist
then=time.time()
for element in tri(100000):
    if is_pentagonal(element)==True and
is_hexagonal(element)==True:
        print(element)
now=time.time()
print(now-then)
```

```
                              1
                             210
                            40755
```

```
7906276
1533776805
0.981999874115 second
```

Problem 41. The great mathematician Euler proposed that every odd (not prime) number can be written as a sum of a prime and double of a square.

$$
\begin{gathered}
9 = 7 + 2\times1^2 \\
15 = 7 + 2\times2^2 \\
21 = 3 + 2\times3^2 \\
25 = 7 + 2\times3^2 \\
27 = 19 + 2\times2^2 \\
33 = 31 + 2\times1^2
\end{gathered}
$$

But the conjecture was not true. Find the least odd number (not prime) which can not be written as the sum of a prime and double of a square?

Solution 41.

```
import eulerlib
import math
def is_sqr(n):
    return math.sqrt(n)==int(math.sqrt(n))
def check(n):
    checklist=[]
    plist=eulerlib.primes(n)
    for p in plist:
        if is_sqr((n-p)/2)==True:
            checklist.append(1)
    if len(checklist)>=1:
        return True
    else: return False
liste=[]
for i in range(2,10000):
    if i%2!=0:
        if eulerlib.is_prime(i)==False:
            if check(i)==False:
                liste.append(i)

print(min(liste))
```

```
5777
```

Problem 42. The numbers 14 and 15 are the first two numbers whose prime factors are different from each other. 644, 645 and 646 are the first three consecutive numbers which have three different prime factors. So, find the first consecutive numbers which have four distinct prime factors.

Solution 42.

```
def prime_factors(n):##12=[2,2,3]
    i=2
    factors=[]
    while i*i<=n:
        if n%i!=0:
            i=i+1
        else:
            n//=i
            factors.append(i)
    if n>1:
        factors.append(n)
    return factors


def unique_prime_factors(n):##84=[2,3,7]
    i=2
    factors=[]
    while i*i<=n:
        if n%i!=0:
            i=i+1
        else:
            n//=i
            if i not in factors:
                factors.append(i)
    if n>1:
        if n not in factors:
            factors.append(n)
    return factors

def findconsprime():
    liste=[i for i in range(3,1000000)]
    for i in liste:
        fournum=[]
        for element in liste[i:i+4]:
            if len(unique_prime_factors(element))==4:
                fournum.append(element)
        if len(fournum)==4:
            break
    return fournum

print(findconsprime())
               [134043, 134044, 134045, 134046]
```

Problem 43. $1^1 + 2^2 + 3^3 + \ldots + 10^{10} = 10405071317$ is given. Calculate the ten digits of the number $1^1 + 2^1 + 3^3 + \ldots + 1000^{1000}$ from right.

Solution 43.

```
def powersum(n):
```

```
    toplam=0
    for i in range(1,n+1):
        toplam=toplam+(i**i)%(10**10)
    return toplam%(10**10)
print(powersum(1000))
                               9110846700
```

Problem 44. The arithmetic sequence, 1487, 4817, 8147, in which each of the terms increases by 3330, is unusual in two ways: (i) each of the three terms are prime, and, (ii) each of the 4-digits numbers are permutations of one another. There are no arithmetic sequences made up of three 1-, 2-, or 3-digits primes, exhibiting this property, but there is one other 4-digits increasing sequence. What 12-digits number do you form by concatenating the three terms in this sequence?

Solution 44.

```
import itertools
import math
import time
def DivNum(a):
    n=int(math.sqrt(a))
    div_list=[]
    for i in range(1, n+1):
        if a%i==0:
            if i and (a//i) not in div_list:
                div_list.append(i)
                div_list.append(a//i)
    return len(div_list)

def is_prime(n):
    if DivNum(n)==2:
        return True
    else:
        return False

def per(n,r):
    L=list(itertools.permutations(n,r))
    newL=[]
    for element in L:
        kelime=''
        for i in range(len(element)):
            kelime=kelime+str(element[i])
        newL.append(kelime)
    return newL
then=time.time()
listeson=[]
for i in range(9000,9999):
    liste=per(str(i),4)
    liste1=[]
    for element in liste:
```

```
        if is_prime(int(element))==True and str(0) not in
element:
            liste1.append(int(element))
    for elem1 in liste1:
        for elem2 in liste1:
            elem3=(elem1+elem2)/2
            if elem3 in liste1 and elem3!=elem2!=elem1:
                if elem1 and elem3 and elem2 not in listeson:
                    listeson.append(elem1)
                    listeson.append(int(elem3))
                    listeson.append(elem2)
print(listeson)
now=time.time()
print(str(now-then)[0:5]+" second")
                         962962992969
                         0.984 second
```

Problem 45. The prime number 41 could be written as the sum of 6 successive prime numbers from 2 to 13. This sum is the longest sum to build a prime number by using consecutive primes less than 100. If we look for the prime number which has the same property under one-thousand is 953 which is the sum of 21 consecutive prime numbers. Find the prime number which is less than one million and can be written as the sum of the most successive prime numbers.

Solution 45.

```
import fkclib
pliste=fkclib.first_n_prime(4000)
sliste=[]
lliste=[]
for i in range(0,len(pliste)):
    for j in range(0,len(pliste)+1):
        toplam=sum(pliste[i:j])
        if toplam not in sliste and fkclib.is_prime(toplam)
and toplam<10**6:
            sliste.append(toplam)
            lliste.append(j-i)
new=dict(zip(sliste, lliste))
print(fkclib.is_prime(max(sliste)))
a=lliste.index(max(lliste))
print(sliste[a])
                            997651
```

Problem 46. If the number 125874 and the number 25174 are taken it is obvious that the second number is equal to two times the first number and they have the same digits but different order. So, find the least positive integer, *n*, such that *2n, 3n, 4n, 5n,* and *6n* includes the same digits.

Solution 46.

```
def kontrol(m,n):
    m=str(m)
    n=str(n)
    liste=[]
    if len(m)==len(n):
        for num in m:
            if m.count(num)==n.count(num):
                liste.append(1)
            else: return False
    if len(liste)==len(m):
        return True
    else: return False

def check(n):
    liste=[n*2, n*3, n*4, n*5, n*6]
    checklist=[]
    for i in range(0,5):
        if kontrol(n, liste[i])==True:
            checklist.append(1)
    if len(checklist)==5:
        return True
    else: return False

import time
once=time.time()
for i in range(100008,10000000,9):
    if check(i)==True:
        print(i)
        break
sonra=time.time()
print(sonra-once)
```

```
                                    142857
                                  0.08 second
```

Problem 47. There are exactly ten ways of selecting three from five, 12345:

123, 124, 125, 134, 135, 145, 234, 235, 245, and 345.

In combinatorics, we use the notation, $^5C_3$ =10. In general, $C\binom{n}{r} = \frac{n!}{r!(n-r)!}$, $where\ r \leq n, n! = n \cdot (n-1) \cdot ... \cdot 3 \cdot 2 \cdot 1$, and 0!=1. It is not until n = 23, that a value exceeds one-million: $^{23}C_{10}$ = 1144066. How many, not necessarily distinct, values of ^nC_r, for $1 \leq n \leq 100$, are greater than one-million?

Solution 47.

```
import time
def fact(n):
    prod=1
    if n==0 or n==1:
        return 1
    else:
        for i in range(1,n+1):
            prod=prod*i
    return prod
def comb(n,r):
    return fact(n)/(fact(r)*fact(n-r))
a=0
for i in range(1,101):
    for j in range(0,i+1):
        b=comb(i,j)
        if b>1000000:
            a=a+1
once=time.time()
print(a)
sonra=time.time()
print("time: {}".format(str(sonra-once)[0:4]))
                                             4075
                                        time: 0.01
```

Problem 48. If we take 47, reverse and add, 47 + 74 = 121, which is palindromic. Not all numbers produce palindromes so quickly. For example,

$$349 + 943 = 1292,$$

$$1292 + 2921 = 4213$$

$$4213 + 3124 = 7337$$

That is, 349 took three iterations to arrive at a palindrome. Although no one has proved it yet, it is thought that some numbers, like 196, never produce a palindrome. A number that never forms a palindrome through the reverse and add process is called a Lychrel number. Due to the theoretical nature of these numbers, and for the purpose of this problem, we shall assume that a number is Lychrel until proven otherwise. In addition you are given that for every number below ten-thousand, it will either (i) become a palindrome in less than fifty iterations, or, (ii) no one, with all the computing power that exists, has managed so far to map it to a palindrome. In fact, 10677 is the first number to be shown to require over fifty iterations before producing a palindrome: 4668731596684224866951378664 (53 iterations, 28-digits). Surprisingly, there are palindromic numbers that are themselves Lychrel numbers; the first example is 4994. How many Lychrel numbers are there below ten-thousand?

Solution 48.

```
import time
start=time.time()
def is_palindrome(n):
    if str(n)[0::]==str(n)[::-1]:
        return True
    else: return False

def counter(n):
    counter=0
    for i in range(1,59):
        a=n+int(str(n)[::-1])
        if is_palindrome(a)==True:
            return counter+1
            break
        else:
            n=a
            counter=counter+1
number=0
for i in range(1,10001):
    if counter(i)==None:
        number=number+1
print(number)
stop=time.time()
print("time:{0}".format(stop-start))
```

```
                              249
                      time:0.547000169754
```

Problem 49. The number 10^{100} is called as a googol. It contains one times 1 and hundred times zeros. If we take the number 100^{100}, it contains one times 1 and two hundred times zeros. The sum of their digits are equal to 1. If we take a natural number as x^y, where x,y<100, find the greatest value of the sum of its digits ?

Solution 49.

```
liste=[]
for i in range(1,100):
    for j in range(1,100):
        a=i**j
        b=str(a)
        toplam=0
        for char in b:
            toplam=toplam+int(char)
            liste.append(toplam)
print(max(liste))
```

```
972
```

Problem 50. It can be shown that the square root of 2 can be written as an infinite continued fraction.

$$\sqrt{2} = 1 + 1/(2 + 1/(2 + 1/(2+\dots))) = 1.414213\dots$$

If we expand this for the first 4 iterations, we will get:

1 + 1/2 = 3/2 = 1.5
1 + 1/(2 + 1/2) = 7/5 = 1.4
1 + 1/(2 + 1/(2 + 1/2)) = 17/12 = 1.41666...
1 + 1/(2 + 1/(2 + 1/(2 + 1/2))) = 41/29 = 1.41379...

99/70, 239/169, and 577/408 are the result of next 3 iterations. But the eighth expansion, 1393/985, is the first example where the number of digits in the numerator exceeds the number of digits in the denominator. In the first one-thousand expansions, how many fractions contain a numerator with more digits than the denominator?

Solution 50.

```
from fractions import Fraction as f

value=f(3,2)
count=0
for i in range(1,1000):
    value=1+f(1,value+1)
    a=str(value).split("/")
    if len(a[0])>len(a[1]):
        count=count+1
print(count)
```

```
153
```

Problem 51. Starting with 1 and spiralling counterclockwise in the following way, a square spiral with side length 7 is formed.

```
37 36 35 34 33 32 31
38 17 16 15 14 13 30
39 18  5  4  3 12 29
40 19  6  1  2 11 28
41 20  7  8  9 10 27
42 21 22 23 24 25 26
43 44 45 46 47 48 49
```

It is interesting to note that the odd squares lie along the bottom right diagonal, but what is more interesting is that 8 out of the 13 numbers lying along both diagonals are prime; that is, a ratio of 8/13 ≈ 62%.

If one complete new layer is wrapped around the spiral above, a square spiral with side length 9 will be formed. If this process is continued, what is the side length of the square spiral for which the ratio of primes along both diagonals first falls below 10%?

Solution 51.

```
import math
import fkclib
liste=[]
a=2
value=1
primecount=0
c=1
side=1
for j in range(1,10**7):
    c=c+4
    side=side+2
    for i in range(1,5):
        value=value+a
        if fkclib.is_prime(value)==True:
            primecount=primecount+1
        if i!=4:
            liste.append(value)
    if primecount/c<1/10:
        print(side, primecount, c, primecount/c)
        break
    a=a+2
             26241 5248 52481 0.09999809454850327
```

Problem 52. The cube, 41063625 (345^3), can be permuted to produce two other cubes: 56623104 (384^3) and 66430125 (405^3). In fact, 41063625 is the smallest cube which has exactly three permutations of its digits which are also cube.

Find the smallest cube for which exactly five permutations of its digits are cube.

Solution 52.

```
liste=[str(i**3) for i in range(1,20000)]
lenliste=[]
```

```
def check(str1,str2):
    checklist=0
    if len(str1)==len(str2):
        for elem in str1:
            if str1.count(elem)==str2.count(elem):
                checklist=checklist+1
        if len(str1)==checklist:
            return True
        else: return False

for eleman1 in liste:
    a=0
    for eleman2 in liste:
        if check(eleman1, eleman2)==True:
            a=a+1
    if a==5:
        print(eleman1)
        break
                             127035954683
```

Problem 53. The 5-digit number, $16807=7^5$, is also a fifth power. Similarly, the 9-digit number, $134217728=8^9$, is a ninth power. How many *n*-digit positive integers exist which are also an *n*th power?

Solution 53.

```
a=0
for i in range(1,100):
    for j in range(1,100):
        if len(str(i**j))==j:
            a=a+1
print(a)
                              49
```

Problem 54. The function φ calculates the number of relatively prime numbers less than any given natural number. For example φ(10)=4 because the relatively prime numbers with 10 are 1, 3, 7, 9. If we look the proportion of the n/ φ(*n*) the maximum value is 3 for the first ten natural numbers (i.e. 6/φ(*6*)=3). For which value of n which is smaller and equal to 10^6, *n*/φ(*n*) has the maximum value?

Solution 54.

```
import math
from numpy import prod

def DivNum(a):
    n=int(math.sqrt(a))
    div_list=[]
```

```
    for i in range(1, n+1):
        if a%i==0:
            if i not in div_list:
                div_list.append(i)
            if a//i not in div_list:
                div_list.append(a//i)
    return len(div_list)
def is_prime(n):
    if n==1:
        return False
    elif DivNum(n)==2:
        return True
    else:s
        return False
def DivNum1(a):
    n=int(math.sqrt(a))
    div_list=[]
    for i in range(1, n+1):
        if a%i==0:
            if i not in div_list:
                if is_prime(i)==True:
                    div_list.append(i)
            if a//i not in div_list:
                if is_prime(a//i)==True:
                    div_list.append(a//i)
    return div_list

def phi(n):
    philist=[]
    for num in DivNum1(n):
        philist.append(1-(1/num))
    return n*(prod(philist))
maxi=0
for i in range(2,1000001):
    a=i/phi(i)
    if a>maxi:
        maxi=a
        index=i
print(maxi)
print(index)
```

510510

4. CONCLUSION AND RECOMMENDATIONS

4.1 Performance

Using Python without some extension modules, SymPy's performance is not as good as other commercial equivalent competitors. But for many applications the general performance of SymPy is sufficient as measured by time or clock cycles, memory occupation, and memory layout. But in some points, we have to accept that the SymPy has some troubles in doing very long expressions or lots of small ones. Indeed, part of the performance problems is due to the OS used, the processor, and other hardware components such as RAM Python's nature as being an interpreted language also brings other performance related issues. During the solution of the problems, many times the author of the thesis had to chance to compare different types of computers and online idles. For example, many problems are solved on the computer which has an Intel R atom processor and some problems are solved on an online Idle repl.it[3]. And the performance difference between a tablet computer and a mini super computer has been obvious. The ratio of the solution times is very high because online Idle was 80 times faster than the pc. So, the boundaries of the software depend on the system, because the modern computers have a range 10^4-10^6 symbols for calculation.

Therefore, a new open source project named SymEngine (The SymPy Developers, 2016) was started. The main aim of this project is to write efficient libraries to make the SymPy has a better performance.

4.2 Conclusion and Future Work

Python language and SymPy support many mathematical facilities. These includes many functions from number theory to calculus. Expression simplifying,

---

[3] www.repl.it

polynomial calculations, pretty printing and using *Miktex*, solving equations, performing symbolic matrices are the most popular functions. Furthermore, plotting 2D and 3D graphs, sets, series, vectors, combinatorics, group theory, cryptography, tensors, code generation, linear algebra can also be counted as special functions. For this reason, many of the users has been choosing SymPy because of its easy usage and free access. When compared with other CAS's SymPy is easy to learn, teach and use since it is being written in pure Python. There are many source to learn Python and SymPy freely. One can also start with the given Python documentation list to explore various features from official site[4]:

- The Statistics Module
- Numeric and Mathematical Modules
- The Math Module
- The Decimal Module (We did not disscuss this module.)
- Floating Point Arithmetic (We did not disscuss this module.)

Beside the official site, one can also explore the mathematics and programming topics from the books:

- Doing Mathematics With Python (Saha, 2015)
- Invent Your Computer Games With Python (Sweigart, 2016)
- Think Stats: Probability and Statistics for Programmers (Downey, 2011)

In addition to all the given internet resources, Project Euler (https://projecteuler.com) is the definite place to take exercises for the ones who would like improve their coding skills. The site includes more than 500 mathematics problems. The problems in the Problems and The Solutions section are selected from this web site. Creating a free account is the only requirement to begin selecting problems to solve and improve thereby coding skills in Python using SymPy.

---

[4] https://docs.python.org

5. REFERRENCES

Barnsley MF (1988) Fractal modeling of real world images. In The science of fractal images, Springer, New York.

Cervone D (2012), “MathJax: A Platform for Mathematics on The Web”, Notices of The AMS, 59(2): 312-316.

Cimrman R (2014) “SfePy-Write Your Own FE Application”, EUroSciPy 2013, August 21-24, Brussels.

Ciurana E (2009) Developing with Google App Engine, 1st Edition, Apress, New York.

Downey AB (2011) Think Stats, v.1.6., Greentea Press, Needham, Massachusetts.

Fu H, Zhong X, Zeng Z (2006) “Automated and Readable Simplification of Trigonometric Expressions”, Mathematical and Computer Modelling, 44(11-12): 1169-1177.

Gede G, Peterson DL, Nanjangud AS, Moore JK and Hubbard M (2013) “Constrained Multibody Dynamics with Python: From Symbolic Equation Generation to Publication”, ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, August 4-7, Portland.

Hénon M (1976) “A two-dimensional mapping with a strange attractor”, The Theory of Chaotic Attractors, 94-102.

Kluyver T, Ragan-Kelley B, Pérez F, Granger B, Bussonnier M, Frederic , Kelley K, Hamrick J, Grout J, Corlay S, Ivanov P, Avila D, Abdalla S, Willing C and Jupyter Development Team (2016) “Jupyter Notebooks—A Publishing Format for Reproducible Computational Workflows”, Positioning and Power in Academic Publishing: Players, Agents and Agendas, 1 0.3233/978-1-61499-649-1-87, 87-90.

Lutz M (2013) Learning Python, O’Reilly Media Inc., Sebastopol.

Oliphant TE (2007) “Python for Scientific Computing”, Computing in Science & Engineering, 9(3): 10-20.

Pérez F and Granger BE (2007) “IPython: a system for interactive scientific computing”, Computing in Science & Engineering, 9(3): 21-29.

Raymond E (1999) “The Cathedral and The Bazaar”, Knowledge, Technology and Policy, 12(3): 23-49.

Roach K (1996) “Hypergeometric function representations”, ISSAC '96 Proceedings of the 1996 international symposium on Symbolic and algebraic computation, 1996, Zurich.

Rosen L (2005) Open Source Licensing: Software Freedom and Intellectual Property Law, Prentince Hall, Upper Saddle River.

Saha A (2015) Doing Math with Python, 1st press, No Starch Press, San Francisco.

Sweigart A (2015) Automate The Boring Stuff with Python, 1st press, No Starch Press, San Francisco.

SymPy Org, Projects Using SymPy, http://www.SymPy.org/en/index.html, 07.04.2017.

The SageMath, The Sage Developers, www.sagemath.org, 07.04.2017.

The SymPy Developers (2016). SymEngine, a fast symbolic manupilation library, written in C++, Available at https://github.com/symengine/symengine.

Project Euler Problems, Archives, https://projecteuler.net/archives, 07.05.2017.

6. CURRICULUM VITAE

Name SURNAME : Fatih Kürşat CANSU

Place and Date of Birth : Kırıkkale, 04.05.1978

Universities

Bachelor's Degree : Abant Izzet Baysal University

MSc Degree : Istanbul University

PhDc Degree : Bahçeşehir University

e-mail : fatihcansu@gmail.com

Address : AİBÜ Lojmanları Lale Blok D:16 Bolu

List of Publications :

1. Cansu F.K. (2010) Ulusal Bilgisayar Olimpiyatları Soru ve Çözümleri 2000-2010, Altın Nokta Yayınları, İzmir.
2. Cansu F.K. (2008) Diyafont Denklemleri Elementer Çözüm Yöntemleri, Altın Nokta Yayınları, İzmir.
3. Cansu F.K. (2012) Matematik Olimpiyat Problemleri ve Çözümleri, Bahçeşehir Üniversitesi Yayınları, İstanbul.