Function

We use a number of common mathematical functions from the Python standard library's math module. Some of the common functions is given below:

function	use
math.acos()	Returns the arc cosine of a number
---	---
math.acosh()	Returns the inverse hyperbolic cosine of a number
---	---
math.asin()	Returns the arc sine of a number
---	---
math.asinh()	Returns the inverse hyperbolic sine of a number
---	---
math.atan()	Returns the arc tangent of a number in radians
---	---
math.atan2()	Returns the arc tangent of y/x in radians
---	---
math.atanh()	Returns the inverse hyperbolic tangent of a number
---	---
math.ceil()	Rounds a number up to the nearest integer
---	---
math.comb()	Returns the number of ways to choose k items from n items without repetition and order
---	---
math.cos()	Returns the cosine of a number
---	---
math.cosh()	Returns the hyperbolic cosine of a number
---	---
math.degrees()	Converts an angle from radians to degrees
---	---
math.dist()	Returns the Euclidean distance between two points (p and q), where p and q are the coordinates of that point
---	---
math.exp()	Returns e raised to the power of x
---	---
math.expm1()	Returns e to the power x - 1
---	---

function	use
math.fabs()	Returns the absolute value of a number
---	---
math.factorial()	Returns the factorial of a number
---	---
math.floor()	Rounds a number down to the nearest integer
---	---
math.fmod()	Returns the remainder of x/y
---	---
math.fsum()	Returns the sum of all items in any iterable (tuples, arrays, lists, etc.)
---	---
math.gamma()	Returns the gamma function at x
---	---
math.gcd()	Returns the greatest common divisor of two integers
---	---
math.hypot()	Returns the Euclidean norm
---	---
math.isqrt()	Rounds a square root number downwards to the nearest integer
---	---
math.log()	Returns the natural logarithm of a number, or the logarithm of number to base
---	---
math.log10()	Returns the base-10 logarithm of x
---	---
math.log1p()	Returns the natural logarithm of 1+x
---	---
math.log2()	Returns the base-2 logarithm of x
---	---
math.perm()	Returns the number of ways to choose k items from n items with order and without repetition
---	---
math.pow()	Returns the value of x to the power of y
---	---
math.prod()	Returns the product of all the elements in an iterable
---	---
math.radians()	Converts a degree value into radians
---	---
math.remainder()	Returns the closest value that can make numerator completely divisible by the denominator
---	---

function	use
math.sin()	Returns the sine of a number
math.sinh()	Returns the hyperbolic sine of a number
math.sqrt()	Returns the square root of a number
math.tan()	Returns the tangent of a number
math.tanh()	Returns the hyperbolic tangent of a number
math.trunc()	Returns the truncated integer parts of a number

In [4]:
```
import math
math.sin(math.pi/2)
```

Out[4]: 1.0

#to install sympy pip install sympy

math.sin() works for numerical values. It gives error ' can't convert expression to float' when it uses symbol. Thus we use the function sympy.sin()

In [6]:
```
#Similar to the standard library's sin() function, SymPy's sin() function expects
import sympy
sympy.sin(math.pi/2)
```

Out[6]: 1.0

1.Derive the expression for the time it takes for a body in projectile motion to reach the highest point if it's thrown with initial velocity u at an angle theta.

In [7]:
```
from sympy import sin, solve, Symbol
u = Symbol('u')
t = Symbol('t')
g = Symbol('g')
theta = Symbol('theta')
solve(u*sin(theta)-g*t, t)
```

Out[7]: [u*sin(theta)/g]

2.Program check whether the expression x + 5 is greater than 0.

In [9]:
```
# define x with sign
x = Symbol('x', positive=True)
# check the condition
if (x+5) > 0:
 print('Do Something')
else:
 print('Do Something else')
```

```
Do Something
```

In [4]:
```
# to get f(y) when f(x) is defined
from sympy import *
y = Symbol('y')
f = lambda x : x**2+2
print(f(y))
```

```
y**2 + 2
```

Addition, Substraction, Multiplication, Division and Composition of Functions

Example. f(x) = $e^x + x^2$

```
    g(x) = 2y+1
```

In [1]:
```
import math
def f(x):
    func_fx=(math.exp(x))+(x * x)
    return func_fx;

def g(y):
    func_gy=(2*y)+1;
    return func_gy;

print("f(1)=",f(1))
print("g(1)=",g(1))

def composite_function_add(f, g):

    return lambda x : (f(x)+g(x))

add_comp=composite_function_add(f,g)
print( "(f+g)(1)=",add_comp(1))

def composite_function_minus(f, g):

    return lambda x : (f(x)-g(x))

minus_comp=composite_function_minus(f,g)
print( "(f-g)(1)=",minus_comp(1))

def composite_function_mult(f, g):

    return lambda x : (f(x)*g(x))

mult_comp=composite_function_mult(f,g)
print( "(fg)(1)=",mult_comp(1))

def composite_function_div(f, g):

    return lambda x : (f(x)/g(x))

div_comp=composite_function_div(f,g)
print( "(f/g)(1)=",div_comp(1))

def composite_function_comp(f, g):

    return lambda x : (f(g(x)))

comp_comp=composite_function_comp(f,g)
print( "(f(g(1))=",comp_comp(1))
```

```
f(1)= 3.718281828459045
g(1)= 3
(f+g)(1)= 6.718281828459045
(f-g)(1)= 0.7182818284590451
(fg)(1)= 11.154845485377136
(f/g)(1)= 1.239427276153015
(f(g(1))= 29.085536923187668
```

```
Task1. f(x) = sin x
```

```
g(x) = $x^2+x$

Determine

1. f+g

2. f-g

3. fg

4. f/g

5. f composition g

6. g composition f

at x=2
```

In [2]:
```
import math
def f(x):
    func_fx=(math.sin(x));
    return func_fx;

def g(x):
    func_gx=x**2+x;
    return func_gx;

print("f(2)=",f(2))
print("g(2)=",g(2))

def composite_function_add(f, g):

    return lambda x : (f(x)+g(x))

add_comp=composite_function_add(f,g)
print( "(f+g)(2)=",add_comp(2))

def composite_function_minus(f, g):

    return lambda x : (f(x)-g(x))

minus_comp=composite_function_minus(f,g)
print( "(f-g)(2)=",minus_comp(2))

def composite_function_mult(f, g):

    return lambda x : (f(x)*g(x))

mult_comp=composite_function_mult(f,g)
print( "(fg)(2)=",mult_comp(2))

def composite_function_div(f, g):

    return lambda x : (f(x)/g(x))

div_comp=composite_function_div(f,g)
print( "(f/g)(2)=",div_comp(2))

def composite_function_comp(f, g):

    return lambda x : (f(g(x)))

comp_comp=composite_function_comp(f,g)
print( "(f(g(2))=",comp_comp(2))
def composite_function_comp1(f, g):

    return lambda x : (g(f(x)))

comp1_comp=composite_function_comp1(f,g)
print( "(g(f(2))=",comp1_comp(2))
```

```
f(2)= 0.9092974268256817
g(2)= 6
```

```
(f+g)(2)= 6.909297426825682
(f-g)(2)= -5.090702573174318
(fg)(2)= 5.45578456095409
(f/g)(2)= 0.1515495711376136
(f(g(2))= -0.27941549819892586
(g(f(2))= 1.7361192372574878
```

Task2. Given

```
f(x) = 3x

g(x) = 2x+5
```

Verify $f(g(x)) \neq g(f(x))$ at all points,

In [4]:

```
import math
def f(x):
    func_fx=3*x;
    return func_fx;

def g(x):
    func_gx=2*x+5;
    return func_gx;

print("f(2)=",f(2))
print("g(2)=",g(2))

def composite_function_comp(f, g):

    return lambda x : (f(g(x)))

comp_comp=composite_function_comp(f,g)
print( "(f(g(2))=",comp_comp(2))
def composite_function_comp1(f, g):

    return lambda x : (g(f(x)))

comp1_comp=composite_function_comp1(f,g)
print( "(g(f(2))=",comp1_comp(2))
if comp_comp(2)!=comp1_comp(2):
    print("f(g(x)) not equal to g(f(x)) at x = 2")
else:
    print("Check for another x")
```

```
f(2)= 6
g(2)= 9
(f(g(2))= 27
(g(f(2))= 17
f(g(x)) not equal to g(f(x)) at x = 2
```

Limits

We can find limits of functions in SymPy by creating objects of the Limit class as follows

In [24]:
```
#  S, which is a special SymPy class that contains the definition of infinity (po
from sympy import Limit, Symbol, S
x = Symbol('x')
m = Limit(1/x, x, S.Infinity)
m
```

Out[24]: $\lim_{x \to \infty} \frac{1}{x}$

In [25]:
```
#To find the value of the limit, we use the doit() method
l = Limit(1/x, x, S.Infinity)
l.doit()
```

Out[25]: 0

3.The prominent mathematician James Bernoulli discovered that as the value of n increases, the term $(1 + 1/n)^n$ approaches the value of e—the constant that we can verify by finding the limit of the function:

In [37]:
```
from sympy import Limit, Symbol, S
n = Symbol('n')
Limit((1+1/n)**n, n, S.Infinity).doit()
```

Out[37]: e

In [1]:
```
from cmath import *
from math import *
from sympy import *
#SymPy is a Python library for symbolic mathematics.
x,y,z = symbols("x,y,z")
```

4.Find $lim_{x-->2}x^2 + 2$

In [2]:
```
limit(x**2+2,x,2)
```

Out[2]: 6

5. Find $lim_{x-->1}x^2 + 2$

In [3]:
```
limit((x**2)+2,x,1)
```

Out[3]: 3

To determine value from LHS and RHS

1) $f(x) = x^2 + 2x + 1$

In [12]:
```
import numpy as np
def limit():
    n=float(input("Enter the value of limit point: "))
    print("z tends to n from left hand side")
    l=np.linspace(n-0.1,n,10)
    f1=l**2+2*l+1
    print(f1)
    print("z tends to n from right hand side")
    m=np.linspace(n,n+0.1,10)
    f2=m**2+2*m+1
    print(f2[::-1])
    print("z at n")
    f3=n**2+2*n+1
    print(f3)
```

In [13]:
```
limit()
```

```
Enter the value of limit point: 5
z tends to n from left hand side
[34.81       34.94123457 35.07271605 35.20444444 35.33641975 35.46864198
 35.60111111 35.73382716 35.86679012 36.        ]
z tends to n from right hand side
[37.21       37.0745679  36.93938272 36.80444444 36.66975309 36.53530864
 36.40111111 36.26716049 36.13345679 36.        ]
z at n
36.0
```

2) $f(x) = 2x + 4$

In [14]:
```
def limit():
    n=float(input("Enter the value of limit point: "))
    print("z tends to n from left hand side")
    l=np.linspace(n-0.1,n,10)
    f1= 2*l + 4
    print(f1)
    print("z tends to n from right hand side")
    m=np.linspace(n,n+0.1,10)
    f2= 2*m + 4
    print(f2[::-1])
    print("z at n")
    f3= 2 *n +4
    print(f3)
```

In [15]:
```
limit()
```

```
Enter the value of limit point: 4
z tends to n from left hand side
[11.8        11.82222222 11.84444444 11.86666667 11.88888889 11.91111111
 11.93333333 11.95555556 11.97777778 12.        ]
z tends to n from right hand side
[12.2        12.17777778 12.15555556 12.13333333 12.11111111 12.08888889
 12.06666667 12.04444444 12.02222222 12.        ]
z at n
12.0
```

In [ ]: