Functions

Python Lambda Functions are anonymous function means that the function is without a name.

1.This function can have any number of arguments but only one expression, which is evaluated and returned.

2.lambda functions can be used in wherever function objects are required.

3.lambda functions are syntactically restricted to a single expression.

In [1]:
```
#to evaluate value of the function at some point x
from sympy import *
f = lambda x : x**2+2
f(2)
```

Out[1]:
```
6
```

In [2]:
```
# 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 [3]:
```
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
```

1. 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
```

2. Given

$$f(x) = 3x$$

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

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

In [5]:
```
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
```

To read function from user

In [2]:
```
import math
import sympy
from sympy import *
x = Symbol('x')
a = input("Enter the first function in terms of x:") #reading function
f=lambda x: eval(a) #function will be stored as an anonymous function to f using
f(x) # lambda function calling using another variable. Because lambda itself canr
```

```
Enter the first function in terms of x:sin(x)
```

Out[2]: $\sin(x)$

3.Write a python program to read functions from user and evaluate it's

a.Sum

b.Difference

c.Product

d.Division

e.Composition

In [4]:
```
import math
import sympy
from sympy import *
x = Symbol('x')
def composite_function_add(f, g):
    return lambda x : (f(x)+g(x))
def composite_function_minus(f, g):
    return lambda x : (f(x)-g(x))
def composite_function_mult(f, g):
    return lambda x : (f(x)*g(x))
def composite_function_div(f, g):
    return lambda x : (f(x)/g(x))
def composite_function_comp(f, g):
    return lambda x : (f(g(x)))
p = input("Enter the first function in terms of x:")
q = input("Enter the second function in terms of x:")
a = lambda x: eval(p)
b = lambda x: eval(q)
add_comp=composite_function_add(a,b)
minus_comp=composite_function_minus(a,b)
multi_comp=composite_function_mult(a,b)
div_comp=composite_function_div(a,b)
comp_comp=composite_function_comp(a,b)
comp_comp2=composite_function_comp(b,a)
print( "(f+g)(x)=",add_comp(x))
print( "(f-g)(x)=",minus_comp(x))
print( "(fg)(x)=",multi_comp(x))
print( "(f/g)(x)=",div_comp(x))
print( "(f(g(x))=",comp_comp(x))
print("g(f(x))",comp_comp2(x))
```

```
Enter the first function in terms of x:2+x
Enter the second function in terms of x:x**2
(f+g)(x)= x**2 + x + 2
(f-g)(x)= -x**2 + x + 2
(fg)(x)= x**2*(x + 2)
(f/g)(x)= (x + 2)/x**2
(f(g(x))= x**2 + 2
g(f(x)) (x + 2)**2
```