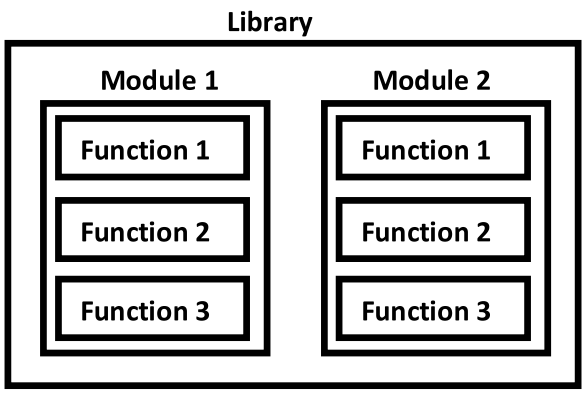
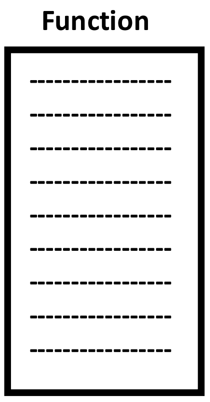
Note: Function vs Module vs Library:

1. A group of lines with some name is called a function
2. A group of functions saved to a file , is called Module
3. A group of Modules is nothing but Library





Types of Variables

Python supports 2 types of variables.

1. Global Variables
2. Local Variables

1. Global Variables

The variables which are declared outside of function are called global variables.
These variables can be accessed in all functions of that module.

Eg:

```
a=10  # global variable
def f1():
    print(a)

def f2():
    print(a)

f1()
f2()

Output
10
10
```

2. Local Variables:

The variables which are declared inside a function are called local variables.
Local variables are available only for the function in which we declared it.i.e from outside of function we cannot access.

Eg:

```
def f1():
    a=10
    print(a)    # valid

def f2():
    print(a)     #invalid

f1()
f2()

NameError: name 'a' is not defined
```

global keyword:

We can use global keyword for the following 2 purposes:

1. To declare global variable inside function
2. To make global variable available to the function so that we can perform required modifications

Eg 1:

```
a=10
def f1():
    a=777
    print(a)

def f2():
    print(a)

f1()
f2()

Output
777
10
```

Eg 2:

```
a=10
def f1():
    global a
    a=777
    print(a)

def f2():
    print(a)

f1()
f2()

Output
777
777
```

Eg 3:

```
def f1():
    a=10
    print(a)

def f2():
    print(a)

f1()
f2()

NameError: name 'a' is not defined
```

Eg 4:

```
def f1():
    global a
    a=10
    print(a)

def f2():
    print(a)

f1()
f2()

Output
10
10
```

Note: If global variable and local variable having the same name then we can access global variable inside a function as follows

```
a=10    #global variable
def f1():
   a=777  #local variable
   print(a)
   print(globals()['a'])
f1()


Output
777
10
```

Recursive Functions

A function that calls itself is known as Recursive Function.

Eg:

```
factorial(3)=3*factorial(2)
             =3*2*factorial(1)
             =3*2*1*factorial(0)
             =3*2*1*1
                   =6
factorial(n)= n*factorial(n-1)
```

The main advantages of recursive functions are:

1. We can reduce length of the code and improves readability
2. We can solve complex problems very easily.

Q. Write a Python Function to find factorial of given number with recursion.

Eg:

```
 def  factorial(n):
   if n==0:
      result=1
   else:
      result=n*factorial(n-1)
   return result
print("Factorial of 4 is :",factorial(4))
print("Factorial of 5 is :",factorial(5))

Output
```

```
Factorial of 4 is : 24
Factorial of 5 is : 120
```

Anonymous Functions:

Sometimes we can declare a function without any name,such type of nameless functions are called anonymous functions or lambda functions.

The main purpose of anonymous function is just for instant use(i.e for one time usage)

Normal Function:

We can define by using def keyword.

```
def  squareIt(n):
    return n*n
```

lambda Function:

We can define by using lambda keyword

```
lambda n:n*n
```

Syntax of lambda Function:

lambda   argument_list : expression

Note: By using Lambda Functions we can write very concise code so that readability of the program will be improved.

Q. Write a program to create a lambda function to find square of given number?

```
s=lambda n:n*n
print("The Square of 4 is :",s(4))
print("The Square of 5 is :",s(5))

Output
The Square of 4 is : 16
The Square of 5 is : 25
```

Q. Lambda function to find sum of 2 given numbers

```
s=lambda a,b:a+b
print("The Sum of 10,20 is:",s(10,20))
print("The Sum of 100,200 is:",s(100,200))

Output
The Sum of 10,20 is: 30
The Sum of 100,200 is: 300
```

Q. Lambda Function to find biggest of given values.

```
 s=lambda a,b:a if a>b else b
print("The Biggest of 10,20 is:",s(10,20))
print("The Biggest of 100,200 is:",s(100,200))

Output
The Biggest of 10,20 is: 20
The Biggest of 100,200 is: 200
```

Note:
Lambda Function internally returns expression value and we are not required to write return statement explicitly.

Note: Sometimes we can pass function as argument to another function. In such cases lambda functions are best choice.

We can use lambda functions very commonly with filter(),map() and reduce() functions,b'z these functions expect function as argument.

filter() function:

We can use filter() function to filter values from the given sequence based on some condition.

filter(function,sequence)

where function argument is responsible to perform conditional check
sequence can be list or tuple or string.

Q. Program to filter only even numbers from the list by using filter() function?

without lambda Function:

```
def isEven(x):
    if x%2==0:
        return True
    else:
        return False
l=[0,5,10,15,20,25,30]
l1=list(filter(isEven,l))
print(l1)  #[0,10,20,30]
```

with lambda Function:

```
l=[0,5,10,15,20,25,30]
l1=list(filter(lambda x:x%2==0,l))
print(l1)   #[0,10,20,30]
l2=list(filter(lambda x:x%2!=0,l))
print(l2) #[5,15,25]
```

map() function:

For every element present in the given sequence,apply some functionality and generate new element with the required modification. For this requirement we should go for map() function.

Eg: For every element present in the list perform double and generate new list of doubles.

Syntax:

map(function,sequence)

The function can be applied on each element of sequence and generates new sequence.

Eg: Without lambda

```
 l=[1,2,3,4,5]
def doubleIt(x):
    return 2*x
l1=list(map(doubleIt,l))
print(l1) #[2, 4, 6, 8, 10]
```

with lambda

```
l=[1,2,3,4,5]
l1=list(map(lambda x:2*x,l))
print(l1)    #[2, 4, 6, 8, 10]
```

Eg 2: To find square of given numbers

```
l=[1,2,3,4,5]
l1=list(map(lambda x:x*x,l))
print(l1)      #[1, 4, 9, 16, 25]
```

We can apply map() function on multiple lists also.But make sure all list should have same length.

Syntax: map(lambda x,y:x*y,l1,l2))
x is from l1 and y is from l2

Eg:

```
l1=[1,2,3,4]
l2=[2,3,4,5]
l3=list(map(lambda x,y:x*y,l1,l2))
print(l3)      #[2, 6, 12, 20]
```

reduce() function:

reduce() function reduces sequence of elements into a single element by applying the specified function.

reduce(function,sequence)

reduce() function present in functools module and hence we should write import statement.

Eg:

```
from functools import *
l=[10,20,30,40,50]
result=reduce(lambda x,y:x+y,l)
print(result) # 150
```

Eg:

```
result=reduce(lambda x,y:x*y,l)
print(result)   #12000000
```

Eg:

```
from functools import *
result=reduce(lambda x,y:x+y,range(1,101))
print(result)   #5050
```