In [1]:

```
num = int(input("Enter the number : "))
s = 0
for i in range(1,num+1):
    s = s + i
print(s)
```

```
Enter the number : 7
28
```

In [4]:

```
def sum_of_num(number):
    s = 0
    for i in range(1,number+1):
        s = s + i
    print(s)
```

In [7]:

```
sum_of_num(15)
```

```
120
```

In [8]:

```
def unique_list(l):
    empty_list = []
    for i in l:
        if i not in empty_list:
            empty_list.append(i)
    print(empty_list)


sanvee = [1,2,3,4,5,1,2,2,2,3,3,4,5]
unique_list(sanvee)
```

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

In [11]:

```
num1 = 23
num2 = 45
if num1>num2 :
    print("num1 is greater")
elif num2 > num1:
    print("num2 is greater")
else:
    print("equal")
```

```
num2 is greater
```

In [12]:

```
def max_min(m,n):
    if m>n :
        print("num1 is greater")
    elif n>m:
        print("num2 is greater")
    else:
        print("equal")
s = max_min(34,67)
print(s)
```

```
num2 is greater
None
```

In [13]:

```
def add(a,b):
    return a+b
add(34,56)
```

Out[13]:

```
90
```

Lambda Function

In Python, an anonymous function is a function that is defined without a name.

While normal functions are defined using the def keyword in Python, anonymous functions are defined using the lambda keyword. Hence, anonymous functions are also called lambda functions.

Syntax of Lambda Function in python

lambda arguments: expression

We use lambda functions when we require a nameless function for a short period of time.

lambda parameter : expression

In [15]:

```
(lambda a,b : a+b)(54,6)
```

Out[15]:

```
60
```

In [16]:

```
add = lambda a,b : a+b
add(56,98)
```

Out[16]:

```
154
```

In [32]:

```
x = lambda a, b : a * b
print(x(5, 6))
```

```
30
```

In [33]:

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

```
13
```

Lambda functions are used along with built-in functions like filter(), map() etc.

The filter() function in Python takes in a function and a list as arguments.

The function is called with all the items in the list and a new list is returned which contains items for which the function evaluates to True.

In [17]:

```
#Example 1: Program for even numbers without lambda function
def even_numbers(nums):
    even_list = []
    for n in nums:
        if n % 2 == 0:
            even_list.append(n)
    return even_list
num_list = [10, 5, 12, 78, 6, 1, 7, 9]
ans = even_numbers(num_list)
print("Even numbers are:", ans)
```

```
Even numbers are: [10, 12, 78, 6]
```

In [18]:

```
#map, filter and reduce
```

In [23]:

```
num_list = [10, 5, 12, 78, 6, 1, 7, 9]
a = list(filter(lambda n : n % 2 == 0, num_list))
print(a)
```

```
[10, 12, 78, 6]
```

In [25]:

```
def square_numbers(nums):
    square_list = []
    for n in nums:
        square_list.append(n*n)
    return square_list
num_list = [10, 5, 12, 78, 6, 1, 7, 9]
ans = square_numbers(num_list)
print("Even numbers are:", ans)
```

```
Even numbers are: [100, 25, 144, 6084, 36, 1, 49, 81]
```

In [26]:

```
num_list = [10, 5, 12, 78, 6, 1, 7, 9]
a = list(map(lambda n : n*n, num_list))
print(a)
```

```
[100, 25, 144, 6084, 36, 1, 49, 81]
```

In [27]:

```
from functools import reduce
numbers = [1, 2, 3, 4, 5]
sum = reduce(lambda x, y: x + y, numbers)
print("Sum:", sum)
```

```
Sum: 15
```

Recursion

A recursive function is a function that calls itself, again and again.

A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.

In [29]:

```
def factorial(no):
     if no == 0:
        return 1
     else:
        return no * factorial(no - 1)
print("factorial of a number is:", factorial(5))
```

```
factorial of a number is: 120
```

In [ ]:

```

```