Functions

In [15]:
```
def num(n):
    if n%2==0:
        return "even"
    else:
        return "odd"
print(num(8))

```

```
even
```

In [12]:
```
def sum(n):
    total=0
    for i in range(1,n+1):
        total=total+i
    return total
n=int(input("enter a number "))
print(sum(n))
#this out works like
#0+1=1
#1+2=3
#3+3=6
#6+4=10
#10+5=15
#15+6=21 etc...

```

```
enter a number 7
28
```

In [42]:
```
#postional argument:
def sum(a,b):
    return a+b
print(sum(4,6))
```

```
10
```

In [43]:
```
#default arguments:
def sum(a,b=90):
    return a+b
print(sum(6))
```

```
96
```

In [46]:
```
#keyword argument:we will pass in the function call
def sum(a,b):
    return a+b
print(sum(a=12,b=13))

```

```
25
```

In [59]:
```
class Calculator:
    def __init__(self,a,b): #default constructor __init__
        self.a=a #self holds the memory we will use self.
        self.b=b
    def add(a,b):
        return a+b
    def sub(a,b):
        return a-b
    def mul(a,b):
        return a*b
    def div(a,b):
        return a/b
obj1=Calculator
obj2=Calculator
obj3=Calculator
obj4=Calculator
print(obj1.add(4,9))
print(obj1.sub(4,9))
print(obj1.mul(4,9))
print(obj1.div(4,9))
```

```
13
-5
36
0.4444444444444444
```

4

```
  abstraction - most top layer whic show the output
  encupl-all the file bundled the data or/ encuplasated.we can achiev
e it by access modifier
      public,private,protected are access modifier
  polymor-many role
```

In [3]:
```
num=int(input("enter number "))
if num>1:
    for i in range(2,int(num/2)+1):
        if(num%i)==0:
            print(num,"is not prime")
            break
    else:
        print(num,"is a prime number")
```

```
enter number 12
12 is not prime
```

code for access modifiers

In [1]:
```
#multiple two parent class and one child class
class father:
    def father_method():
        return "i'm father method"

class mother:
    def mother_method():
        return "i'm mother method"

class child(father,mother):
    def child_method():
        return "i'm child method derived from both father and mother"
obj1=child
print(obj1.child_method())
print(obj1.father_method())
print(obj1.mother_method())

```

```
i'm child method derived from both father and mother
i'm father method
i'm mother method
```

In [3]:
```
#inheritance
class animal:
    def animal_level_1_method():
        return "im animal method"
class dog(animal):
    def dog_level_2_method():
        return "im dog method,im inherited from class animal"
class puppy(dog):
    def puppy_level_3_method():
        return"im puppy method,inherited from class dog"
obj1=puppy
print(obj1.animal_level_1_method())
print(obj1.dog_level_2_method())
print(obj1.puppy_level_3_method())
```

```
im animal method
im dog method,im inherited from class animal
im puppy method,inherited from class dog
```

In [12]:
```
class mammel:
    def mammal_level_0_method():
        return"im mammal method"
class animal(mammel):
    def animal_level_1_method():
        return "im animal method"
class dog(animal,mammel):
    def dog_level_2_method():
        return "im dog method,im inherited from class animal"
class puppy(dog):
    def puppy_level_3_method():
        return"im puppy method,inherited from class dog"
obj1=puppy
print(obj1.mammal_level_0_method())
print(obj1.animal_level_1_method())
print(obj1.dog_level_2_method())
print(obj1.puppy_level_3_method())
```

```
im mammal method
im animal method
im dog method,im inherited from class animal
im puppy method,inherited from class dog
```

method overriding

In [3]:
```
class Animal:
    def speak():
        return "Animal is speaking"
class Dog:
    def speak():
        return"dog is barking"
class Cat:
    def speak():
        return"meow"
animal=Animal
dog=Dog
cat=Cat
print(animal.speak())
print(dog.speak())
print(cat.speak())
```

```
Animal is speaking
dog is barking
meow
```

In [10]:
```
#overloading and variable length
class Calculator:
    def add(self,*args):
        total=0
        for num in args:
            total +=num
        return total
calculator=Calculator()
print(calculator.add(2,3))
print(calculator.add(2,3,4))
print(calculator.add(2,3,4,5))
print(calculator.add(2,3,4,5,6))
```

```
5
9
14
20
```

In [11]:
```
#prime
num=int(input("enter number "))
if num>1:
    for i in range(2,int(num/2)+1):
        if(num%i)==0:
            print(num,"is not prime")
            break
    else:
        print(num,"is a prime number")
```

```
enter number 11
11 is a prime number
```

In [3]:
```
num=int(input("enter number "))
if num%2==0:
    print(num,"even")
else:
    print(num,"odd")
```

```
enter number 12
12 even
```

parameters and arguments diff

Parameters:1.Parameters are variables declared in a function or method definition.
2.They act as placeholders for the values that will be passed into the function when it is called.
3.Parameters define the input that a function expects.
4.Parameters are part of the function signature.

Arguments:
1.Arguments are the actual values that are passed to a function when it is called.
2.They correspond to the parameters defined in the function's declaration.
3.Arguments are the concrete values that are supplied to a function for processing.
4.They are provided when calling the function.

In [1]:
```
def is_palindrome(s):

    s = s.replace(" ", "").lower()

    return s == s[::-1]


input_string = input("Enter a string: ")


if is_palindrome(input_string):
    print(f'"{input_string}" is a palindrome.')
else:
    print(f'"{input_string}" is not a palindrome.')

```

```
Enter a string: malayalam
"malayalam" is a palindrome.
```