In [ ]:
```
#Why we need Function?
#Resuability of code
```

In [8]:
```
x=20
y=30
z=x+y
print(z)
```
```
50
```

In [6]:
```
x=20
y=40
z=x+y
print(z)
```
```
60
```

In [7]:
```
x=60
y=40
z=x+y
print(z)
```
```
100
```

In [ ]:
```
#What are Functions
Functions are the block of code that we use to perform a specific task or logic
#BENEFITS OF FUNCTIONS:
REUSABILITY OF CODE
```

In [19]:
```
def add(x,y):   #x,y are formal parameters that are used while defining the function
    return x+y
print(add(10,20))  # 10,20, are known as actual parameters
print(add(100,200))
print(add(2000,300))
print(add(60,40))
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [19], in <cell line: 3>()
      1 def add(x,y):
      2     return x+y
----> 3 print(add(10))
      4 print(add(100,200))
      5 print(add(2000,300))

TypeError: add() missing 1 required positional argument: 'y'
```

In [ ]:
```
#How many types of Functions we have
1.Builtin Functions  -->functions that are already defined by the python virtual machine-->
id() , type(),print(),len()
#2.User Defined Functions --> functions which are developed by the programmer
#according to business requriement
```

In [13]:
```
#Builtin Function function
x=10
print(id(x))
print((typex))
print(x)
```
```
1382265481808
<class 'int'>
10
```

In [ ]:
```
#Syntax of User Defined Functions
def function_name(parameters):
    return None

# while creating a function we should use two keyword:
1.def() -->def is mandatory
2.return  --> return is optional
```

In [18]:
```
#Print hello world with the help of function.

def wish():
    print("Good Morning")
    return "Good Morning"


print(wish())
#Note: if we are not giving any return statment then PVM will automatically return None.
```
```
Good Morning
Good Morning
```

In [ ]:

In [ ]:
```
#Parameters of Functions?
Parameters are the inputs for the functions based and that parameters our function will work
if we are giving any parameters while defining the function then it is very xompulsory to give the values for that
pararmeters while calling that function

Two types of parameters:
    1.Actual Parameters
    2.Formal Parameter
```

In [30]:
```
#Python function that will print squre of the number? I want to add 15 in it
def squareit(number):
    print("Hello world")

x=squareit("Hello")
print(x)
```
```
Hello world
None
```

In [28]:
```
#Python function that will print squre of the number? I want to add 15 in it
def squareit(number):
    return number**2
x=squareit(5)
x+15
```
Out[28]:
```
40
```

In [ ]:
```
#Return Statement:
functions can take inputs in form of parameters and execute business logic and return output
for the caller function with return Statement
```

In [40]:
```
#odd even
def even_odd(number):
    if number%2==0:
        return " It is an even Number" , "even number","Congrats"
    else:
        return "It is an odd Number"
number=int(input())
x=even_odd(number)
x
```
```
10
```
Out[40]:
```
(' It is an even Number', 'even number', 'Congrats')
```

In [ ]:
```
#odd even
def even_odd(number):
    if number%2==0:
        return " It is an even Number" , "even number","Congrats"
    else:
        return "It is an odd Number"
number=int(input())
x=even_odd(number)
x
```

In [ ]:
```
Note: We can return more than one arguments at a type
```

In [4]:
```
x=[(),(10,20,30),(10),(3,4)]
for i in x:
     if len(i)==0:
            x.remove(i)
print(x)
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [4], in <cell line: 2>()
      1 x=[(),(10,20,30),(10),(3,4)]
      2 for i in x:
----> 3      if len(i)==0:
      4             x.remove(i)
      5 print(x)

TypeError: object of type 'int' has no len()
```

In [45]:
```
def Calculator(x,y):#x,y are formal parameters that are used while defining the function
    add=x+y
    sub=x-y
    mul=x*y
    div=x/y
    mod=x%y
    return add , sub , mul,div ,mod
x= Calculator(10,20)
for i in x:
    print("Calculator details are",i)
```
```
Calculator details are 30
Calculator details are -10
Calculator details are 200
Calculator details are 0.5
Calculator details are 10
```

In [ ]:
```
Types of Paramaters:
    1.Formal --> while defining the function
    2.Actual --> while calling the function
```

In [ ]:
```
Actual Parameters are also divided into 4 types:
    1.Positional Argument
    2.Keyword Argument
    3.Default Argument
    4.Variable length argument
```

In [ ]:
```
#Positional Argument
these are the arguments passed to the function with the correct positional order.
```

In [46]:
```
def add(x,y):
    return y-x
print(add(10,20))
```
```
10
```

In [49]:
```
#Keyword Argument
def add(x,y):
    return y-x
print(add(x=10,y=20))
#In keyword argument all formal parameters value is given by the keys at the caller function
```
```
10
```

In [56]:
```
#Default Argument --> sometimes we can provide default balues for our positional arguments
def wish(name=1232):
    print("Hello "+str(name)+"How are you")
wish(99887)
```
```
Hello 99887How are you
```

In [ ]:
```
#Variable Length Argument
if we don't know how many argument we need to pass while calling a function
then we should use variable length argument
```

In [65]:
```
#In variable length argument python will automatcailly consider parameter as a tuple.
def sums(*n):
    #print(type(n))
    return max(n)

#print(sums()) #-->0
print(sums(10))  #-->10
print(sums(20,30))  #-->50
print(sums(40,50,60))  #-->150
print(sums(50,60,70,30,20,40,50,60,30,40,50,50,304,50,503,20,302,402,0))
print(sums("Max","Min"))
```
```
10
30
60
503
Min
```

In [ ]: