Example Function with default argument or optional argument

```
def sort(a:list,reverse=False):
    if reverse==False:
        for i in range(len(a)):
            for j in range(len(a)-1):
                if a[j]>a[j+1]:
                    a[j],a[j+1]=a[j+1],a[j]
    elif reverse==True:
        for i in range(len(a)):
            for j in range(len(a)-1):
                if a[j]<a[j+1]:
                    a[j],a[j+1]=a[j+1],a[j]


list1=[7,3,6,1,5,2,4,8,10,9]
print(f'before sorting {list1}')
sort(list1)
print(f'after sorting {list1}')
sort(list1,reverse=True)
print(f'after sorting {list1}')
```

Output

```
before sorting [7, 3, 6, 1, 5, 2, 4, 8, 10, 9]
after sorting [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
after sorting [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Python does not support pass by value, when ever function is called by sending object, PVM does not send object but it send reference/address of object.

Example:

```
def swap(a:list,p1=None,p2=None):
    if p1==None or p2==None:
        print("Cannot Swap or interchange values")
    else:
        a[p1],a[p2]=a[p2],a[p1]
```

```
def main():
    list1=[10,20,30,40,50]
    print(f'Before Swaping {list1}')
    swap(list1,0,-1)
    print(f'After Swaping {list1}')


main()
```

Output:

```
Before Swaping [10, 20, 30, 40, 50]
After Swaping [50, 20, 30, 40, 10]
```

Example:

```
def simple_interest(amt,t,r=1.5):
    si=(amt*t*r)/100
    return si


def  main():
    res1=simple_interest(45000,12)
    res2=simple_interest(50000,24,2.0)
    print(res1)
    print(res2)

main()
```

Output:

```
8100.0
24000.0
```

Function with variable length arguments

Variable length argument or parameters receives 0 or more values.
Variable length argument is prefix with *
Variable length argument is of type tuple.
A function is defined with one variable length argument.
When more than one function performs similar operation on different number of values, use function with variable length argument.

Syntax:

```
def <function-name>(*arg-name):
        statement-1
        statement-2
```

Example:

```
def fun1(*a):
    print(a)




fun1()
fun1(10)
fun1(10,20,30)
fun1(1,2,3,4,5,6,7,8,9,10)
```

Output:

```
()
(10,)
(10, 20, 30)
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

Packing and unpacking of collection

What is unpacking in Python?
The process of splitting the packed values into individual elements is called ' unpacking '. The packed values are strings, lists, tuples, sets, and dictionaries . During unpacking the elements from RHS (Right Hand Side) are split into their relative position on the LHS (Left Hand Side).

We use two operators * (for tuples) and ** (for dictionaries) for packing and unpacking.

Example:

```
>>> a,b,c=(10,20,30)
>>> print(a,b,c)
10 20 30
>>> t1=(10,20,30,40,50)
>>> a,b=t1
Traceback (most recent call last):
  File "C:\Program Files\JetBrains\PyCharm Community Edition 2023.2.1\plugins\python-ce\helpers\pydev\pydevconsole.py", line 364, in runcode
    coro = func()
           ^^^^^^
  File "<input>", line 1, in <module>
ValueError: too many values to unpack (expected 2)
>>> a,b,*c=t1
>>> print(a,b,c)
10 20 [30, 40, 50]
>>> list1=[10,20,30,40,50]
>>> a,b,c,d,e=list1
>>> print(a,b,c,d,e)
10 20 30 40 50
>>> a,b,c,*d=list1
>>> print(a,b,c,d)
10 20 30 [40, 50]
```

Example

```
def maximum(*values):
    if len(values)==0:
        return None
    if len(values)==1:
        return values[0]
    else:
```

```
    m=values[0]
    for value in values:
        if value>m:
            m=value
    return m


res1=maximum()
print(res1)
res2=maximum(10)
print(res2)
res3=maximum(10,50,20,60,30,40)
print(res3)
```

Output:

```
None
10
60
```