# function with default/optional argument

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

def main():
    si1=simple_interest(10000,12,2.0)
    si2=simple_interest(5000,16)
    print(si1,si2,sep="\n")

main()
```

Output:

```
2400.0
1200.0
```

Variable length arguments

An argument which receive more than one value is called variable length argument

Variable length argument is prefix with *

Variable length argument is type of tuple

Variable length argument receive 0 or more

If function required more than input to perform operation use variable length arguments

A function is defined with only one variable length argument

~~def max_two(a,b):~~
~~if a>b:~~
~~return a~~
~~else:~~
~~return b~~

res=max_two(10,20)

~~def max_three(a,b,c):~~
~~if a>b and a>c:~~
~~return a~~
~~elif b>c:~~
~~return b~~
~~else:~~
~~return c~~

~~res=max_three~~(10,30,20)

def maximum(*a):

maximum(10,20)
maximum(10,20,30)
maximum(10,20,30,40)

Example:

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

def main():
    fun1()
    fun1(10,20)
    fun1(10,20,30,40,50)
    fun1(10,"python",1.5,1+2j)
main()
```

Output:

```
() <class 'tuple'>
(10, 20) <class 'tuple'>
(10, 20, 30, 40, 50) <class 'tuple'>
(10, 'python', 1.5, (1+2j)) <class 'tuple'>
>>>
```

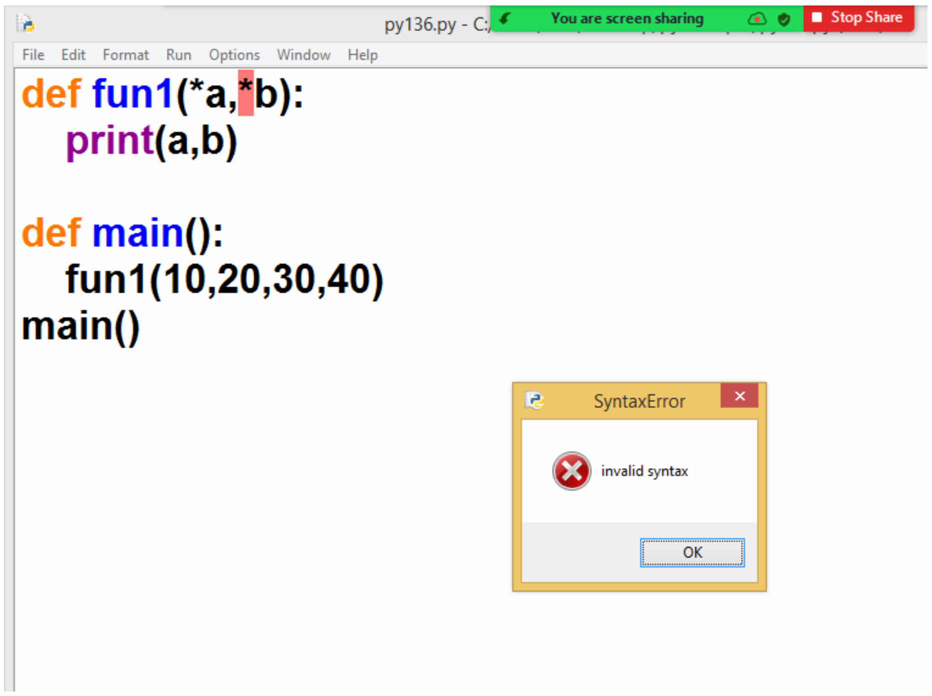
Example:

```
# find max of n numbers

def maximum(*a):
    m=0
    for value in a:
        if value>m:
            m=value
    return m

def main():
    res1=maximum(10,20)
    res2=maximum(30,10,40,50,20)
    print(res1,res2)

main()
```

Output:

```
20 50
```

A function is defined with only one variable length argument



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

def main():
    fun1(10,20,30,40)
main()
```

```
invalid syntax
```

```
def fun1(a,*b,c=10):
    print(a,b,c)
def fun2(a,c=10,*b):
    print(a,b,c)

def main():
    fun1(100)
    fun1(100,200)
    fun1(100,200,300)
    fun1(100,200,300,400,500,c=900)
    fun2(100)
    fun2(100,200,300)
    fun2(1000,*b=10,20,30) # error
main()
```

The order of defining arguments are

1. Required arguments
2. Variable length arguments
3. Keyword arguments
4. Default arguments

Keyword arguments

The function required input as key and value is defined with keyword arguments

Keyword arguments are prefix with **
Keyword argument is of type dictionary
Keyword arguments are used to perform two operations

1. Invoking function by sending key and value
2. For manipulating dictionaries

Example:

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

def main():
    fun1()
    fun1(x=100)
    fun1(x=10,y=20,z=30)

main()
```

Output:

```
{} <class 'dict'>
{'x': 100} <class 'dict'>
{'x': 10, 'y': 20, 'z': 30} <class 'dict'>
>>>
```

Example:

```
def add(**k):
    total=0
    for key,value in k.items():
        total=total+value
        print(f'{key}--->{value}')
    print(f'Total is {total}')

def main():
    add(x=10,y=20)

main()
```

Output:

```
x--->10
y--->20
Total is 30
>>>
```

Example:

```
def add(*a,**k):
    total=0
    if len(a)!=0:
        for value in a:
            print(value)
            total=total+value

    if len(k)!=0:
        for key,value in k.items():
            total=total+value
            print(f'{key}--->{value}')


    print(f'Total is {total}')

def main():
    add(100,200,300,400,500)
    add(x=10,y=20)
    add(100,200,300,x=10,y=20)
main()
```

Output:

```
100
200
300
400
500
Total is 1500
x--->10
y--->20
Total is 30
100
200
300
x--->10
y--->20
Total is 630
```

Example:

```
def display(**k):
    tot=0
    for year,sales in k.items():
        print(f'{year}--->{sales}')
        tot=tot+sales
    print(f'Total Sales {tot}')

def main():
    sales_dict={'2000':450000,'2001':540000,'2002':560000}
    display(**sales_dict) # dictionary unpacking


main()
```

Output:

```
2000--->450000
2001--->540000
2002--->560000
Total Sales 1550000
>>>
```