return keyword
return is a passes control statement or branching statement
return is a keyword
return statement, return value from called function to calling function.
After returning value, it terminates execution of function

Syntax:
return [expression/value]

Example:
```
def power(n,p):
    r=n**p
    return r

def main(): #calling function
    res=power(5,2)
    print(f'result is {res}')
main()
```

Output:
result is 25

Example:
```
def fun1():
    return 10,20,30

def main():
    t=fun1()
    print(t)
    print(type(t))

main()
```
Output:
```
(10, 20, 30)
<class 'tuple'>
>>>
```

Example:
```
def add(a,b):
    return a+b
```

```
def main():
    x=int(input("enter first value"))
    y=int(input("enter second value"))
    z=add(x,y)
    print(f'sum of {x} and {y} is {z}')

main()
```

Output:

```
enter first value5
enter second value6
sum of 5 and 6 is 11
>>>
```

Pass by reference or pass by address

Python supports only pass by reference, whenever function is called by sending objects, python does not send objects but send address of those objects.

Example:

```
def sort(s):
    for i in range(len(s)):
        for j in range(0,len(s)-1):
            if s[j]>s[j+1]:
                s[j],s[j+1]=s[j+1],s[j]

def main():
    list1=[5,3,8,2,4,9,1,7]
    print(f'Before sorting {list1}')
    sort(list1)
    print(f'After sorting {list1}')

main()
```

Output:

```
Before sorting [5, 3, 8, 2, 4, 9, 1, 7]
After sorting [1, 2, 3, 4, 5, 7, 8, 9]
>>>
```

Default arguments or optional arguments

Default arguments/optional arguments are given values at the time of defining function. At the time of calling or invoking function if the values are not given for default arguments, it assign default values.

Syntax:
def <function-name>(arg1,arg2=value,arg3=value,…):
statement-1
statement-2

Example:

```
def draw_line(ch,l=30):
    for i in range(l):
        print(ch,end='')
    print()

def sort(s,order="ascending"):
    if order=="ascending":
        for i in range(len(s)):
            for j in range(len(s)-1):
                if s[j]>s[j+1]:
                    s[j],s[j+1]=s[j+1],s[j]
    elif order=="descending":
        for i in range(len(s)):
            for j in range(len(s)-1):
                if s[j]<s[j+1]:
                    s[j],s[j+1]=s[j+1],s[j]

def main():
    draw_line('*')
    draw_line('$',50)
    list1=[4,7,9,2,4,5,1,6,12,11]
    sort(list1)
    print(list1)
    sort(list1,"descending")
    print(list1)
main()
```

Output:

```
******************************
```

$$
[1, 2, 4, 4, 5, 6, 7, 9, 11, 12]
[12, 11, 9, 7, 6, 5, 4, 4, 2, 1]
>>>

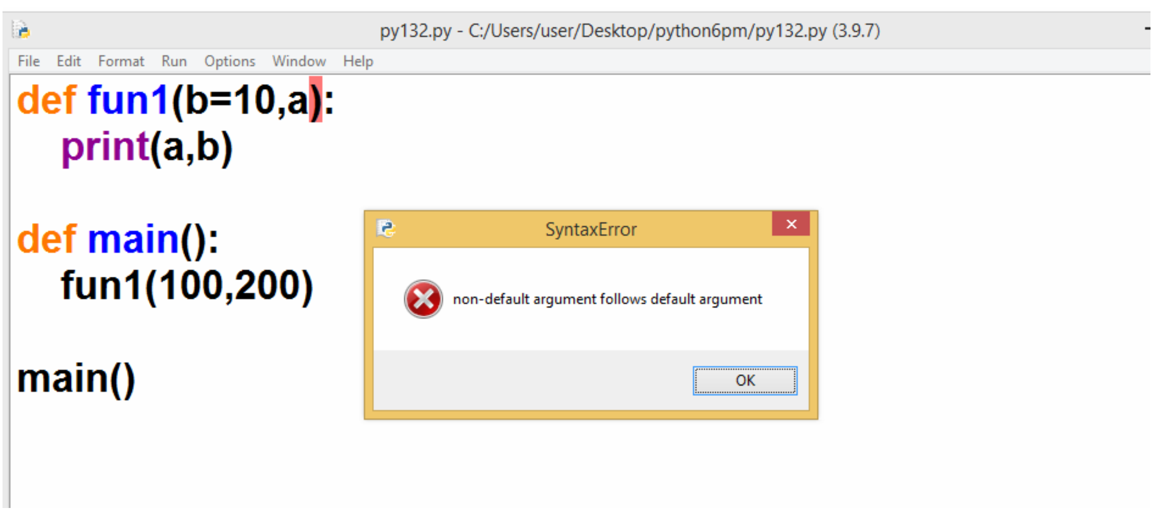
Example:

```
def sum_list(l):
    t=0
    for value in l:
        t+=value
    return t
def main():
    list1=[10,20,30,40,50]
    s=sum_list(list1)
    print(s)

main()
```

Output:

```
150
>>>
```



The above code generate syntax error because default arguments followed by non default arguments