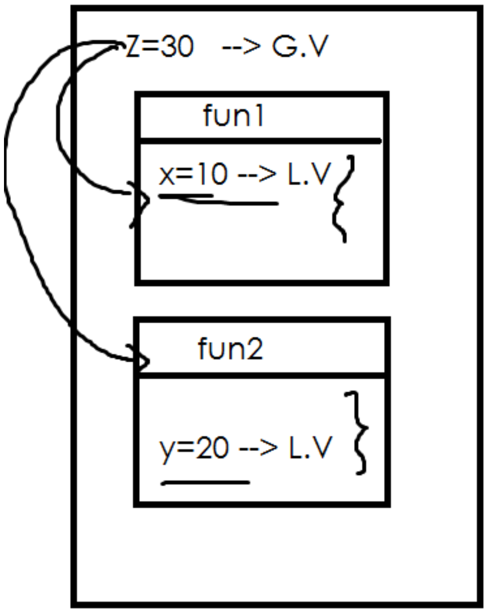
Global variables
Variable declared outside the function are called global variables.
Global variables are global to one or more than one function.



Example:

```
x=100 # Global variable
y=200 # Global variable

def fun1():
    print(x)
    print(y)

def fun2():
    print(x)
    print(y)

fun1()
fun2()
```

Output:

```
100
200
100
200
```

Example:
```
x=100 # Global Variable
def fun1():
   x=200 # L.V
   print(x)

def fun2():
   print(x)


fun1()
fun2()
```

Output:
```
200
100
```

global keyword

The global statement is a declaration which holds for the entire current code block. It means that the listed identifiers are to be interpreted as globals. It would be impossible to assign to a global variable without global, although free variables may refer to globals without being declared global.

Syntax: global variable-name, variable-name, variable-name

Example:
```
x=100 # global variable
def fun1():
   global x
   x=200 # Global Variable
   y=300 # Local variable
   print(x,y)

def fun2():
   print(x)

fun1()
fun2()
```

Output:
```
200 300
200
```

Example:
```
base=0.0
height=0.0

def read():
    global base,height
    base=float(input("Enter Base"))
    height=float(input("Enter Height"))

def find_area():
    area=0.5*base*height
    print(f'Area of triangle is {area:.2f}')


read()
find_area()
```

Output:
```
Enter Base1.2
Enter Height1.5
Area of triangle is 0.90
```

globals()
Return the dictionary implementing the current module namespace

Example:
```
x=100 # Global variable
y=200 # Global variable

d=globals()
print(d)
```

Output:
```
{'__name__': '__main__', '__doc__': None, '__package__': None,
'__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__spec__': None,
'__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, '__file__':
'F:/python6pmaug/funtest10.py', 'x': 100, 'y': 200, 'd': {...}}
```

Example:

```
x=100 # global variable
def fun1():
    x=200 # local variable
    print(x)
    d=globals()
    print(d['x'])
    d['x']=300


fun1()
print(x)
```

Output:

```
200
100
300
```

What is difference between local variable and global variable?

Local variable	**Global variable**
A variable declared inside function is called local variable	A variable declared outside the function is called global variable
This variable is accessed within function (OR) scope of this variable inside function	The scope of these variable within module or outside module
Local variables memory is allocated within function	Global variables memory is allocated as part of module
Memory is allocated when function is called and de-allocated after execution of function	Memory is allocated when module is executed and de-allocated after execution of module.

Example:

```
n1=int(input("enter n1 value")) # global variable
n2=int(input("enter n2 value")) # global variable

def add():
    print(f'sum is {n1+n2}')
```

```
def sub():
    print(f'diff is {n1-n2}')
def multiply():
    print(f'product is {n1*n2}')
def div():
    print(f'result is {n1/n2}')


add()
sub()
multiply()
div()
```

Output:

```
enter n1 value5
enter n2 value2
sum is 7
diff is 3
product is 10
result is 2.5
```

Function with arguments

Function with arguments receives values at the time of invoking or calling the function (OR) if function required input to perform operations we define that function with arguments.

Python allows to define function with 4 types of arguments/parameters

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

Arguments are local variables for which memory is allocated when function is called and de-allocated after execution of function.

Required position arguments or Required arguments

Required argument required value at the time calling or invoking function. if the value of these argument is not given or send python virtual generate error.

Syntax:

```
def <function-name>(arg1,arg2,arg3,….):
        statement-1
        statement-2
        statement-2
```

Example:

```
# function with required positional arguments

def fun1(a,b,c): # function with 3 arguments
    print(a,b,c,sep="\n")



fun1(10,20,30)
fun1("java","python","django")
fun1(1.5,2.5,3.5)
fun1(100,200,300)
fun1(c=1.5,a=100,b=1+2j)
```

Output:

```
10
20
30
java
python
django
1.5
2.5
3.5
100
200
```

```
300
100
(1+2j)
1.5
```

Example:

```
def string_length(s):
    c=0
    for ch in s:
        c+=1
    print(f'length of string is {c}')



str1=input("enter any string") # nit
string_length(str1)
a=len(str1)
print(f"length of string is {a}")
```

Output:

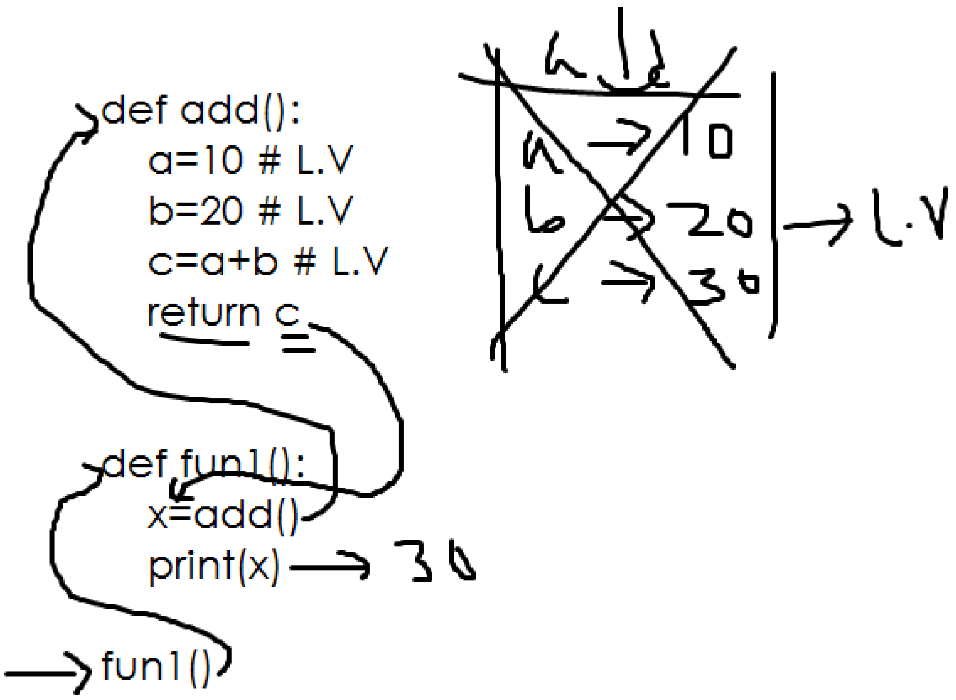
```
enter any stringnit
length of string is 3
length of string is 3
```

return keyword

return keyword or passes control statement
this statement or keyword is used to return value to calling function.

Syntax: return <value>/<expression>

After returning value, return keyword terminates execution of function.



```
def add():
    a=10 # L.V
    b=20 # L.V
    c=a+b # L.V
    return c

def fun1():
    x=add()
    print(x)

fun1()
```

Example:

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



num=int(input("enter any number"))
p=int(input("enter p value"))
res=power(num,p)
print(res)
```

Output:

```
enter any number2
enter p value3
8
```

Whenever function returns value, it is assigned one variable (OR) function is assigned to variable, if function returns value.

Example:

```
def vowel_count(s):
    c=0
    for ch in s:
        if ch in "aeiouAEIOU":
            c+=1
    return c



str1=input("enter any string") # java
k=vowel_count(str1)
print(f'Vowel count is {k}')
```

Output:

```
enter any stringpython
Vowel count is 1
```

Default arguments or Optional arguments