globals()
Return the dictionary implementing the current module namespace. For code within functions, this is set when the function is defined and remains the same regardless of where the function is called.

Example:

```
x=100 # Global Variable
def fun1():
   x=300 # Local Variable
   print(x)
   a=globals()
   print(a)
   print(a['x'])

def fun2():
    x=500
    print(x)
    g=globals()
    g['x']=900
    print(g['x'])
```

Output:

```
 300
{'__name__': '__main__', '__doc__': None, '__package__': None,
'__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__spec__':
None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>,
'__file__': 'E:/python7amdec23/funtest14.py', 'x': 100, 'fun1': <function
fun1 at 0x000001D94A222CA0>, 'fun2': <function fun2 at
0x000001D94A222C00>}
100
500
900
```

```
900


fun1()
fun2()
print(x)
```

global keyword perform two operations

1. Modifying global variable value within function
2. Creating global variable within function

Example:

```
x=100
def fun1():
    print(x) # accessing global variable

def fun2():
    global x
    x=500 # modifing global variable

def fun3():
    global y
    y=200 # creating global variable


fun1()
fun2()
fun1()
fun3()
print(y)
```

Output

```
100
500
200
```

Function with parameters or arguments

Function parameters or arguments are called local variables, these variables are used within function but cannot accessible outside the function.

Parameters/arguments are receivers, which receive values at the time of calling function.

Syntax:

```
def <function-name>(param1,param2,param3,…):
        statement-1
        statement-2
```

Python allows write function with 4 types of arguments or parameters

1. Required arguments or parameters (OR) Required positional argument
2. Default arguments or parameters
3. Variable length arguments or parameters
4. Keyword arguments or parameters

Required arguments or Required positional arguments
Function with required arguments, required values at the time of invoking the function or calling function.
To required arguments not given any values, it leads to syntax error.

Example:
```
def fun1(a,b): # function with 2 arguments (Required)
    print(a,b)



fun1(100,200)
fun1(b=500,a=300)
fun1(1.5,2.5)
fun1(1+2j,2+3j)
fun1("python","django")
fun1([10,20,30,40],1.5)
```

Output:
```
100 200
300 500
1.5 2.5
(1+2j) (2+3j)
python django
[10, 20, 30, 40] 1.5
```

The functions defined in python are generic functions; these functions can receive values of any type.

Example:
```
# write a function to multiply two integers
def multiply(a,b):
    if isinstance(a,(int,float)) and isinstance(b,(int,float)):
        print(a*b)
    else:
        print("please input integer or float values")

def add(a:int,b:int): # type hint
    print(a+b)




multiply(5,2)
multiply("abc","xyz")
multiply(1.5,2.5)
add(10,20)
add(1.5,2.5)
```

Output:

```
10
please input integer or float values
3.75
30
4.0
```

return keyword

"return" passes control statement or branching statement or keyword.
A function return value to caller or calling function using return keyword.
Return keyword, after returning value it terminates execution of function.
Whenever function returns value, caller must assign that function to one variable.

Syntax: return [expression/value/object]

Example:
```
def is_even(num):
    if num%2==0:
        print(True)
    else:
        print(False)

def is_odd(num):
    if num%2!=0:
        return True
    else:
        return False


is_even(4)
res=is_odd(7)
if res:
    print("odd")
else:
    print("even")
```

Output:
```
True
Odd
```

Example:
```
import math
def fun1():
    print("python")
    return
    print("java")

def fun2():
    return 10
    return 20
    return 30

def fun3():
    return (10,20,30)


fun1()
a=fun2()
print(a)
t=fun3()
print(t)
res=math.sqrt(9)
print(res)
```

Output:

```
python
10
(10, 20, 30)
3.0
```