Local Variables

The variables created inside function are called local variables; these variables are created within function memory/function context.

The scope of these variables within function and lifetime of these variables are until execution of function.

Syntax:

```
def  function-name([parameters]):
     local-variable-name=value
     local-variable-name=value
```

Example:

```
def fun1():
   x=100 #Local Variable
   y=200 #Local Variable
   print(x,y)

def fun2():
   print(x,y) # Error

fun1()
fun2()
```

Output

```
100 200
Traceback (most recent call last):
  File "D:/python11amfasttrack/test232.py", line 10, in <module>
    fun2()
  File "D:/python11amfasttrack/test232.py", line 7, in fun2
    print(x,y)
NameError: name 'x' is not defined
```

Global variables

The variables created outside the function are called global variables; these variables can be accessible within same function and outside functions.
In application development in order to share data between number of functions we declare global variables.

Syntax:

```
global-variable-name=value
global-variable-name=value
def <function-name>([parameters]):
        local-variable-name=value
        local-variable-name=value
```

Example:

```
x=100 #Global Variable
def fun1():
   print(x)
def fun2():
   print(x)
def fun3():
   print(x)

fun1()
fun2()
fun3()
```

Output

```
100
100
100
```

Example:

```
n1=10 # Global Variable
n2=5 # Global Variable
def add():
   print(f'sum of {n1} and {n2} is {n1+n2}')
```

```
def sub():
    print(f'diff of {n1} and {n2} is {n1-n2}')

def multiply():
    print(f'product of {n1} and {n2} is {n1*n2}')

def div():
    print(f'division of {n1} and {n2} is {n1/n2:.2f}')


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

Output

```
sum of 10 and 5 is 15
diff of 10 and 5 is 5
product of 10 and 5 is 50
division of 10 and 5 is 2.00
```

Example:

```
x=100 #Global Variable
def fun1():
    print(x)

def fun2():
    x=200 # Local Variable
    print(x)

fun1()
fun2()
fun1()
```

Output

```
100
200
```

Points to remember

1. A function can access global variable directly
2. A function cannot assign value to global variable directly. If function is trying to assign value, it creates variable locally.

global keyword

"global" keyword is used inside function to perform 2 operations.

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

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

after this statement, list variable names are referred as global.

Example:

```
x=100
def fun1():
   print(x)
def fun2():
   global x
   x=200
def fun3():
   global y
   y=500
def fun4():
   print(y)

fun1()
fun2()
fun1()
fun3()
fun4()
```

Output
100
200
500

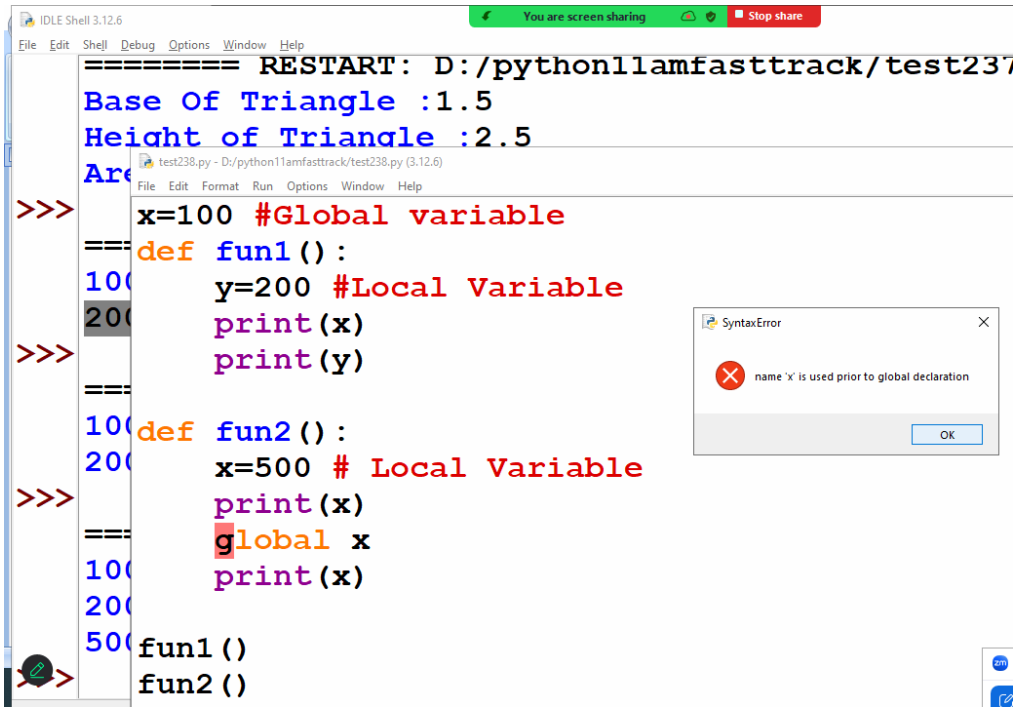
Example:
```
# To Find Area of Triangle
def read():
    global base,height
    base=float(input("Base Of Triangle :"))
    height=float(input("Height of Triangle :"))

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

read()
find_area()
```

Output
Base Of Triangle :1.5
Height of Triangle :2.5
Area of triangle is 1.88

Example:



The above code generates syntax error because the list variable name in global keyword should not be used before global keyword.

globals()

globals() is an in-built function in python. This function returns a dictionary contains global identifiers or names exist in current module or program.

Syntax: variable-name=globals()

Example:

```
x=100 # global variable
y=200 # global variable
def fun1():
   a=globals()# create dictionary
   print(a)
   for k,v in a.items():
      print(f'{k}----->{v}')

fun1()
```

Output

```
{'__name__': '__main__', '__doc__': None, '__package__': None,
'__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__spec__':
None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>,
'__file__': 'D:/python11amfasttrack/test239.py', 'x': 100, 'y': 200, 'fun1':
<function fun1 at 0x000002124D364900>}
__name__----->__main__
__doc__----->None
__package__----->None
__loader__----->><class '_frozen_importlib.BuiltinImporter'>
__spec__----->None
__annotations__----->{}
__builtins__----->><module 'builtins' (built-in)>
__file__----->D:/python11amfasttrack/test239.py
x----->100
y----->200
fun1----->><function fun1 at 0x000002124D364900>
```

Example:

```
x=100 # Global Varaible
def fun1():
    x=200 # Local Variable
    print(x)
    a=globals()
    print(a['x'])
    a['x']=500

fun1()
print(x)
```

Output

```
200
100
500
```

Function with parameters or arguments

Function without parameters cannot receive values (OR) caller cannot give any input.
Function with parameters receives values (OR) caller can give input.
In application development a function is defined with parameters, if function required input to perform operation.

Python allows writing function with 3 types of parameters or arguments

1. Function with required parameters or arguments
    a. Function with required positional only parameters
    b. Function with required keyword only parameters
2. Function with default parameters or optional parameters
3. Function with variable length parameters
    a. Function with variable length positional parameters
    b. Function with variable length keyword parameters

Function with required parameters or arguments
Function with required parameters required values at the time of invoking function or calling function.

Syntax:

```
def function-name(param-name,param-name,param-name,..):
        statement-1
        statement-2
```

Note: parameter names are local variables.

Example:

```
def simple_interest(amt,t,rate):
   si=(amt*t*rate)/100
   print(f'Simple Interest is {si:.2f}')


simple_interest(8000,12,1.5)
```

Output
Simple Interest is 1440.00

Example:

```
def fun1(a,b,c,d):
    print(a,b,c,d)



fun1(10,20,30,40)
fun1(a=100,b=200,c=300,d=400)
fun1(c=600,a=300,d=700,b=900)
```

Output
10 20 30 40
100 200 300 400
300 900 600 700

Function with required parameters or argument is invoked or called in 2 ways

1. By sending values using parameter position (positional argument)
2. By send values using parameter name (keyword arguments)