Local variables

The variables created inside function are called local variables. These variables are used within function but cannot accessible outside the 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()
#print(x,y) Error
#fun2()
```

Output

```
100 200
```

Example:

```
def add():
   n1=int(input("Number1 :"))
   n2=int(input("Number2 :"))
   n3=n1+n2
   print(f'Sum of {n1} and {n2} is {n3}')

add()
```

Output

```
Number1 :10
Number2 :5
```

Sum of 10 and 5 is 15

Global Variables
A variable created outside the function is called global variable
These variables are global to program, these variables can be used within same function and outside function.
In application development in order to share data between more than one function it is declared as global.
Global variables can shared between programs.

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
y=200 # Global Variable

def fun1():
   print(x,y)

def fun2():
   print(x,y)

fun1()
fun2()
```

Output
100 200
100 200
Local variables are deleted automatically after execution of function
Global variables are deleted automatically after execution of program or application.

Example:

```
n1=int(input("First Number :")) # Global Variable
```

```
n2=int(input("Second Number :")) # Global Variable

def add():
    n3=n1+n2 # Local Variable
    print(f'sum of {n1} and {n2} is {n3}')

def sub():
    n3=n1-n2 # Local variable
    print(f'diff of {n1} an {n2} is {n3}')

add()
sub()
```

Output

```
First Number :5
Second Number :2
sum of 5 and 2 is 7
diff of 5 an 2 is 3
```

A function can access global variable directly but cannot update or modify value of global variable directly.

Example:

```
x=100 # Global Variable

def fun1():
    print(x)

def fun2():
    x=500 # Local Variable
    print(x)


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

Output

```
100
500
```

“global” keyword

Syntax:
global identifier-name,identifier-name,…

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.

Names listed in a global statement must not be used in the same code block textually preceding that global statement

Example:

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

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

def fun2():
    global x,y
    x=500
    y=600

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

Output

```
100
200
500
600
```

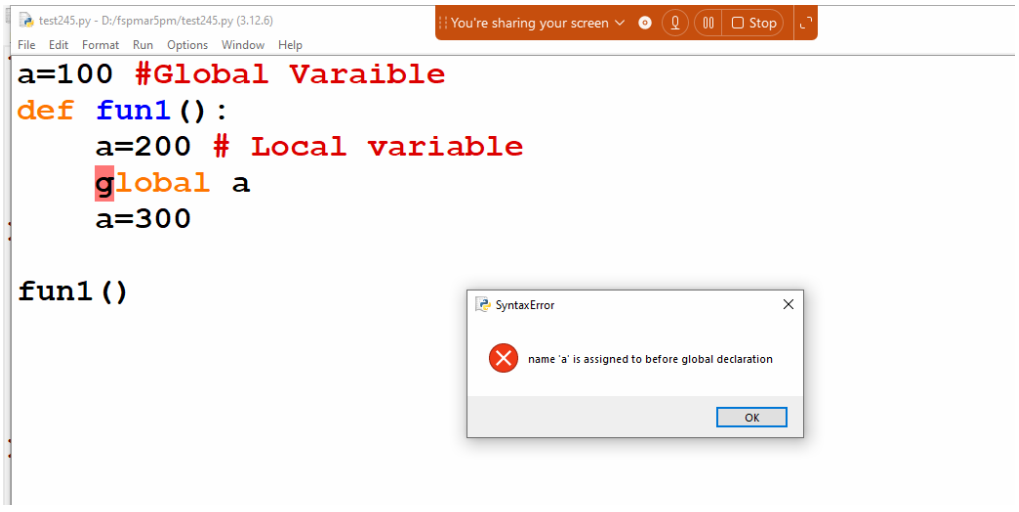
Example:

```
# Find area of triangle

base=0.0  # G.V
height=0.0 # G.V
def read():
    global base,height
    base=float(input("Base :"))
    height=float(input("Height :"))

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

read()
find_area()
```

Output

```
Base :1.5
Height :1.2
Area is 0.90
```

Names listed in a global statement must not be used in the same code block textually preceding that global statement



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.
Globals() function returns global names, python maintain global names in one dictionary. The reference of this dictionary can be read using globals function

Example:

```
x=100
y=200
z=300
def fun1():
    pass
def fun2():
    pass
def fun3():
    pass

g=globals()
print(g)
```

Output

```
{'__name__': '__main__', '__doc__': None, '__package__': None,
'__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__spec__': None,
'__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, '__file__':
'D:/fspmar5pm/test246.py', 'x': 100, 'y': 200, 'z': 300, 'fun1': <function fun1 at
0x00000153C3964900>, 'fun2': <function fun2 at 0x00000153C3964720>,
'fun3': <function fun3 at 0x00000153C39649A0>, 'g': {...}}
```

Example:

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

def fun1():
    x=400
    y=600
    print(f'Local variable x={x}')
    print(f'Local variable y={y}')
    g=globals()
    print(f"Global variable x={g['x']}")
```

```
    print(f"Global varaible y={g['y']}")
    g['x']=1000
    g['y']=2000

fun1()
print(f'global variable x={x}')
print(f'global variable y={y}')
```

Output

```
Local variable x=400
Local variable y=600
Global variable x=100
Global varaible y=200
global variable x=1000
global variable y=2000
```

Function with parameters

What is parameter?

Parameter is a local variable which receives value from caller
A function with parameter receives values at the time of invoking or calling the function.
In application development a function is defined with parameter, if required input to perform some operation.

Python allows defining function with different types of parameters

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

Function with required arguments or parameters

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

Syntax:

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

Example:

```
def fun1(a,b): # required parameter
    print(a,b)


fun1(100,200) # required positional arguments
fun1(b=300,a=400) # required keyword arguments
fun1(1.5,2.5)
fun1("python",3.13)
```

Output

```
100 200
400 300
1.5 2.5
python 3.13
```